Análisis e implementación de un jugador automático de 2048

Autor:

Javier Álvarez-Campana Carracedo

Tutor:

Carlos Linares López

“Toca hacer el trabajo que se tiene delante”

Ronda de noche, Terry Pratchett.

Agradecimientos

Hay mucha gente a la que tengo que agradecer las fuerzas y los ánimos que me han dado para conseguir este proyecto.

A mi madre, sin cuyas llamadas de preocupación y de ánimo no me hubiese visto para continuar. A mi padre, por todas las horas invertidas en escucharme y ayudarme a revisar el documento. Y a ambos, por todos los esfuerzos y sacrificios que han hecho por darme lo que quería y ayudarme a ser la persona que soy hoy en día. Gracias de verdad.

A Mónica, por esa pizza en el momento justo. Por la continua compañía, por los abrazos de ánimo y por ser simplemente maravillosa. Si nos perdemos, nos veremos en la luna.

A mis amigos, que me han dado las risas necesarias para coger fuerzas. A Marcos y a Víctor, porque el Marvel siempre estará allí y nuestros barcos nunca se oxidarán. A Javi (lok'tar, cole'a) y a Delia, por acogerme como a uno más cuando me perdía en la capital. A Alfredo y a Natalia, por sus consejos informáticos y por ayudarme con la memoria. Y a todos ellos por, de una forma u otra, estar siempre allí.

Por último, a Carlos, sin cuyo apoyo y conocimientos no habría podido acabar este proyecto.

Una vez más, a todos muchísimas gracias.

Resumen

Este proyecto pretende analizar e implementar un jugador automático del famoso puzle 2048. Este juego consiste en juntar piezas que van emergiendo en el tablero para formar otras con un valor superior. El objetivo último del puzle es conseguir la mayor pieza posible, siendo el primer objetivo alcanzar una con el valor 2048.

Se ha comenzado realizando un análisis de los anteriores intentos para resolver el juego. Se determinó que un buen punto de partida sería establecer una media de jugadores humanos medios, que tienen una puntuación media de aproximadamente 20.000 puntos.

A continuación, se realizó el agente que resolvería el puzle utilizando árboles de búsqueda deterministas y no deterministas. Además, se implementó una interfaz gráfica que permitirá al jugador o diseñador utilizarlo de manera más cómoda.

Se ejecutaron un gran número de pruebas para determinar cuál es la mejor configuración para la resolución del juego. Se obtuvieron bastantes métodos para conseguir mejores puntuaciones que jugadores humanos, siendo el método determinista “α-β” el mejor de todos.

Por último, el proyecto finaliza con la comparación de los objetivos alcanzados con los pretendidos, así como una enumeración de líneas futuras, conclusiones finales, planificación del proyecto y un resumen extenso en inglés.

Abstract

This project aims to analyze and implement an automatic player of the famous puzzle 2048. This game consists to put together pieces that are emerging on the board to form other with a higher value. The ultimate objective of the puzzle is to get the biggest piece possible, being the first goal to reach one with value 2048.

It has begun making an analysis of previous attempts to solve the game. It was determined that a good starting point would be to establish an average human players media, which have an average score of approximately 20,000 points.

Then performed the agent that would solve the puzzle using deterministic and non-deterministic search trees. In addition, a graphical interface that will allow the player or designer to use it more comfortably has implemented.

A large number of tests were carried out to determine the best settings for the game resolution. Many methods get better scores than human players, being the deterministic method "α-β" the best of all

Finally, the paper finish with a comparison of the intended objectives achieved with the intended, as well as a list of future lines, final conclusions, project planning and an extensive summary in English.

Índice

Índice de ilustraciones

Índice de tablas

Glosario y definiciones

Ficha (pieza): cada uno de los posibles valores que puede haber en cada una de las casillas del tablero.

Función de evaluación: conjunto de distintas estrategias que se utilizan para resolver problemas.

Lista asociada: una lista que contiene las casillas libres de un tablero.

Movimiento absurdo: el que, una vez realizado, no difiere en nada al tablero original. Cuando todos los movimientos son absurdos, el juego ha terminado.

Tablero: posición del juego. Contiene el valor de cada casilla.

Algoritmo determinista: aquel que, bajo un mismo escenario, siempre obtiene el mismo resultado.

Algoritmo no determinista: aquel que puede obtener distintos resultados ante una misma situación.

Capítulo 1
Introducción

Con el avance de los dispositivos móviles, cada vez más personas aportan sus ideas de aplicaciones de forma pública. Cogen su proyecto, lo desarrollan y lo ponen en el mercado virtual, esperando que tenga tanto éxito como ellos esperan.

Unos ejemplos clarísimos de este fenómeno son los juegos "*Flappy Bird*", el "*2048*", "*Angry birds*", "Limbo", entre otros. Todos ellos tienen, en mayor o menor medida, dos características comunes.

La primera es su sencillez. Los juegos o aplicaciones complicadas no atraen al gran público, puesto que es la propia dificultad lo que repercute en la falta de diversión.

La segunda es su facilidad para atrapar la atención. Cuando una persona ve a un amigo jugando a uno de estos juegos, le llama la atención. Enseguida se baja la aplicación, lo que supone horas y horas muertas pasando el rato con ella.

Gabriele Cirulli era un joven con una idea. Se trataba de un juego matemático que consiste en ir juntando piezas hasta lograr una del valor 2048. Así nació el "2048", uno de los juegos más importantes de esta plataforma de los últimos años.

El juego funciona del siguiente modo. Se comienza con un tablero de 4x4 con dos fichas en lugares aleatorios. Cuando se realiza un movimiento, las fichas se mueven y se juntan, formando piezas más grandes (esto se explicará de forma más extensa en el capítulo 2).

Ilustración 1 Tablero de ejemplo del juego 2048

A pesar de su apariencia sencilla, no es en absoluto trivial. Mucha gente se queda atascada sin ser capaz de conseguir grandes puntuaciones, lo cual frustra y engancha a la vez. Es por esto que se pretende hacer un análisis del juego, así como un programa que utilice Inteligencia Artificial para resolver el juego de la mejor forma posible.

La idea y motivación de este proyecto surge de las múltiples horas que me he pasado jugando a este puzle. En medio de una partida, me pregunté si me sería posible crear un jugador automático que explote las distintas estrategias que sigo y que fuera capaz de jugar mejor que yo.

Para ello, se irán desarrollando los distintos conceptos en capítulos de la siguiente memoria:

Capítulo 2 Estado del arte: Para entender mejor cómo funciona la Inteligencia Artificial en este tipo de juegos, se explicarán los algoritmos utilizados, así como otra información de utilidad.

Capítulo 3 Objetivos: Constará de una lista con los objetivos de este trabajo.

Capítulo 4 Desarrollo: En este apartado, se irán desglosando cada uno de los pasos que se han desarrollado en este proyecto.

Capítulo 5 Pruebas y resultados: Con el desarrollo acabado, se pasará a ejecutar una serie de pruebas para comprobar si los objetivos del capítulo 3 se han logrado.

Capítulo 6 Conclusiones y líneas futuras: Se concluye comparando los resultados con los objetivos. De forma adicional, se describe una serie de mejoras que se pueden incluir en el futuro.

Capítulo 7 Planificación y presupuesto: En este capítulo se hablará de la planificación de este proyecto, los plazos cumplidos, así como el presupuesto y los costes del mismo.

Capítulo 2
Estado de la cuestión

En este capítulo se explicarán las tecnologías utilizadas para el desarrollo de este proyecto, así como el uso de las mismas. También se presentará con más profundidad el juego “2048”.

2.1. 2048

Como se ha explicado en el capítulo anterior, el “2048” es un juego para plataformas móviles cuyo objetivo es juntar fichas de un valor idéntico con el fin de obtener una ficha nueva con el doble de valor. Con ese proceso, el juego plantea el reto de alcanzar una pieza de valor 2048 o más.

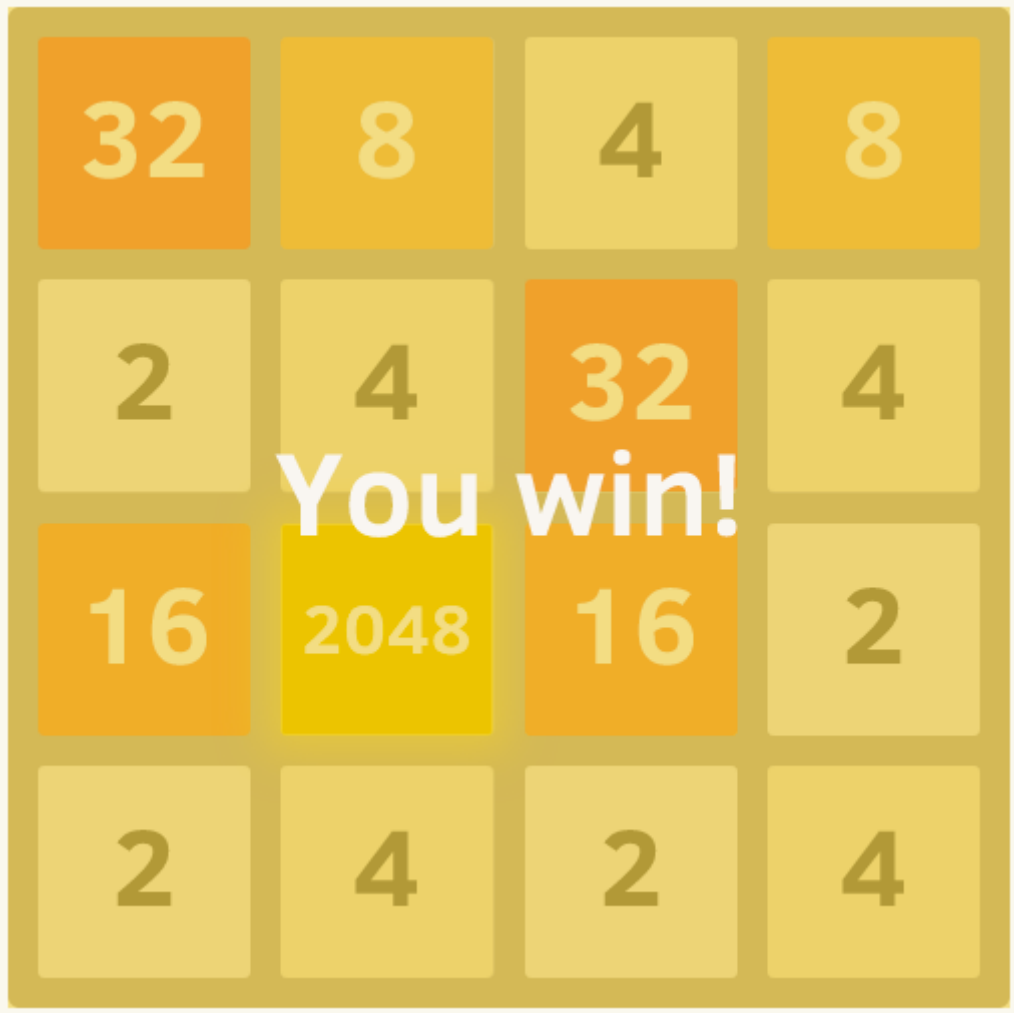


Ilustración 2 Ejemplo de victoria "2048"

En el enlace [6] se podrá encontrar la página original del juego.

2.2. Reglas 2048

Como ya se ha mencionado anteriormente, el 2048 es un juego del tipo *slide-and-merge* (deslizar y juntar), lo que significa que la dinámica del juego consiste en realizar un movimiento de deslizamiento a lo que el propio juego responderá haciendo aparecer una ficha nueva.

Las reglas de este juego son muy sencillas. Se comienza con un tablero, por defecto con 4x4 casillas, en el que hay dos casillas 2 distribuidas de forma aleatoria.

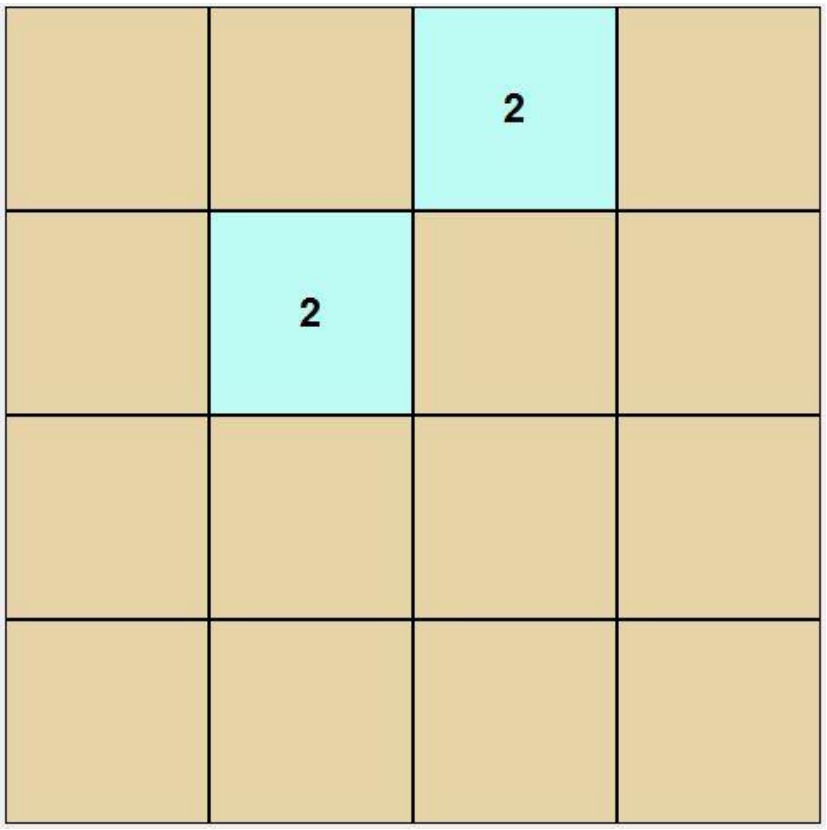


Ilustración 3 Ejemplo de tablero inicial

Con el tablero ya iniciado, podremos desplazar todas las casillas a una determinada dirección, ya sea arriba, abajo, izquierda o derecha. En este caso, estos son los posibles movimientos.

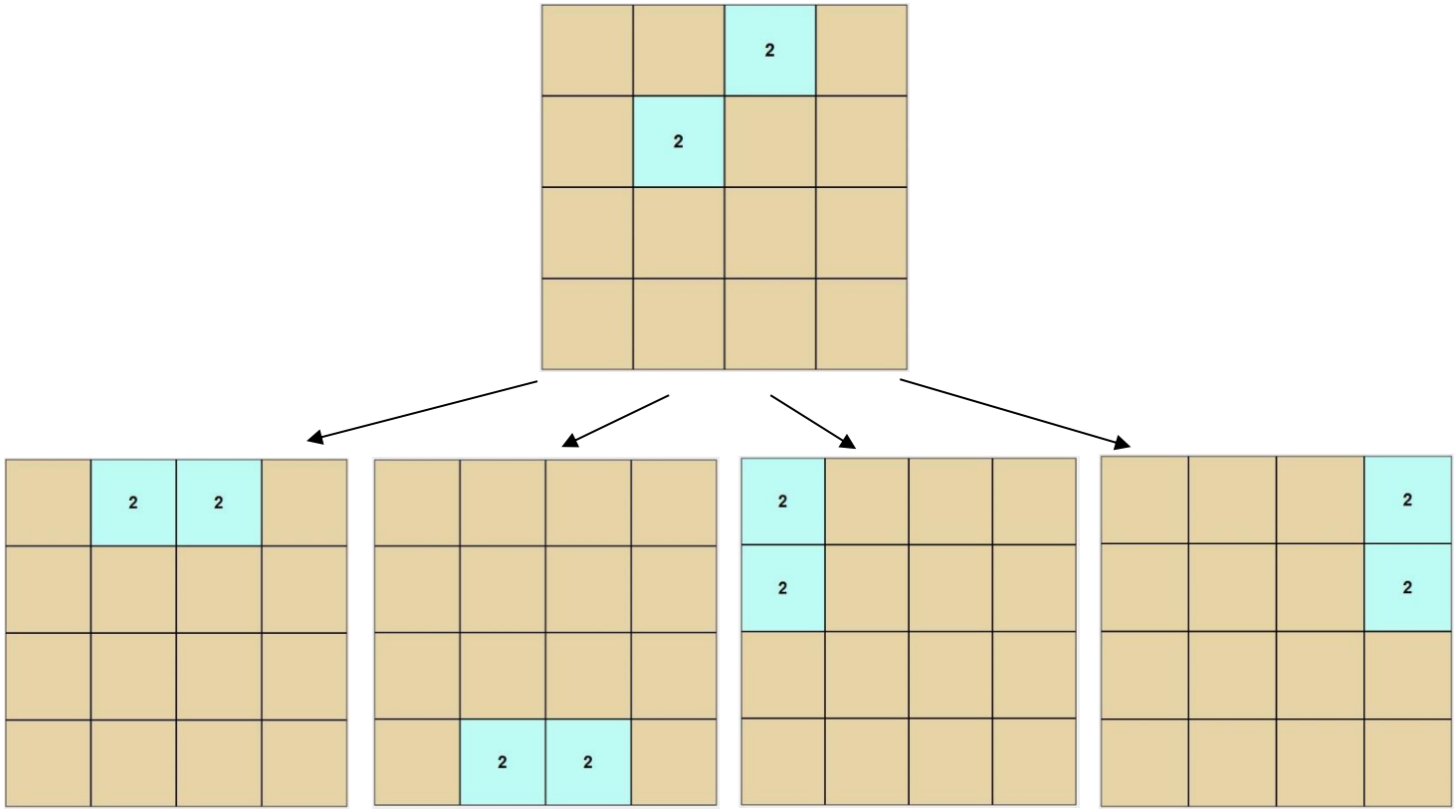


Ilustración 4 Ejemplo de movimientos

A continuación, el juego coloca otra ficha 2 en alguna casilla vacía aleatoria. Continuando con el ejemplo anterior, si se hubiese realizado el movimiento de derecha, el tablero podría quedar como se muestra a continuación:

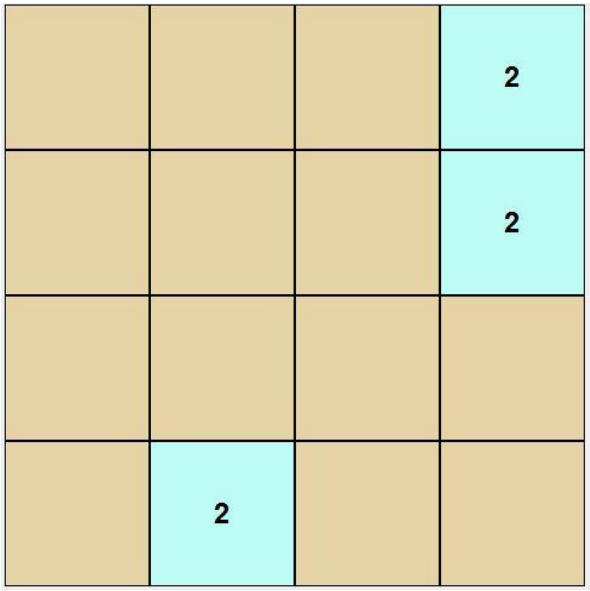


Ilustración 5 Ejemplo de resultado de la primera jugada

Sin embargo, en ciertas ocasiones existen algunos movimientos, a partir de ahora llamados “movimientos absurdos”, que no se pueden realizar por no producir ningún cambio. Por ejemplo, el siguiente tablero no es posible realizar el movimiento “arriba”:

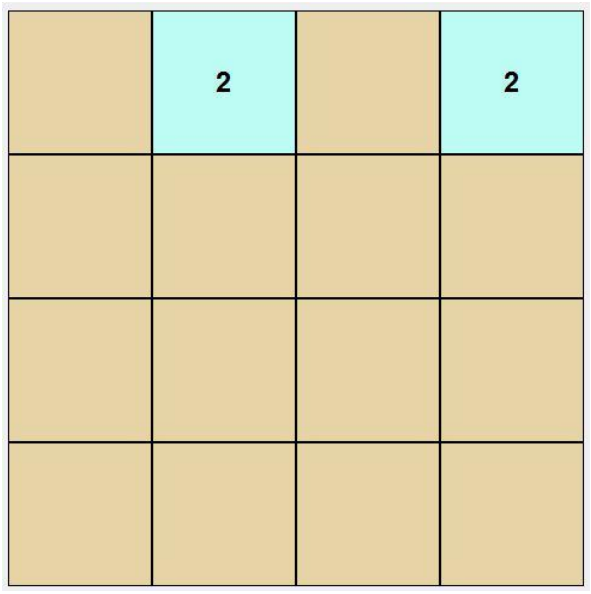


Ilustración 6 Ejemplo de movimiento absurdo

Como ya se ha comentado, el objetivo del juego consiste en juntar piezas para obtener una mayor. Por ejemplo, en el anterior tablero realizamos un movimiento derecha, con el siguiente resultado (incluyendo la introducción de una ficha aleatoria nueva):

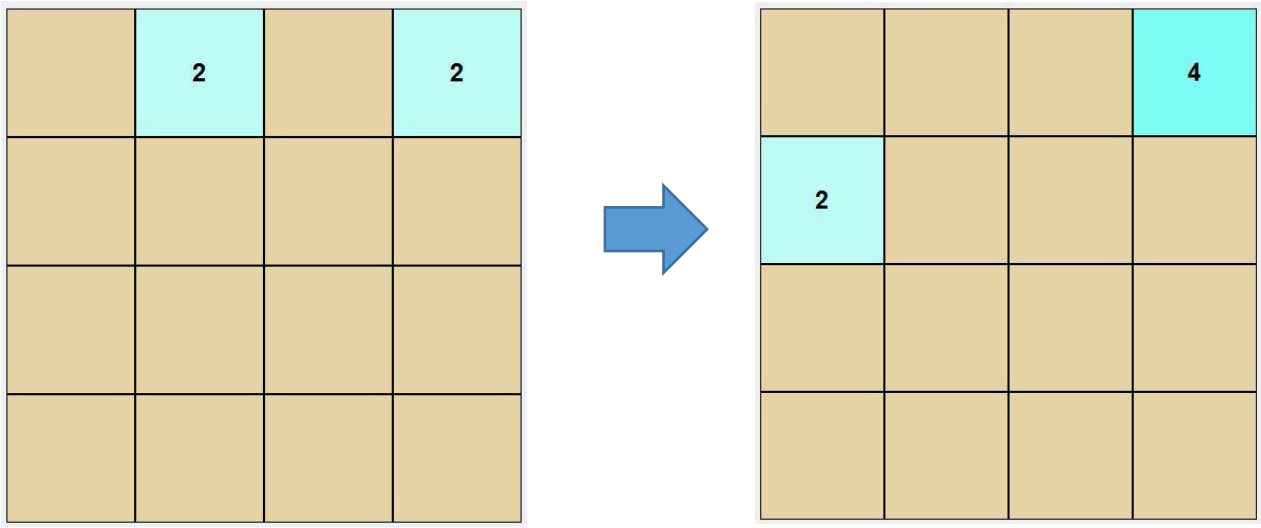


Ilustración 7 Ejemplo de juntar piezas

Como se puede apreciar, se han juntado los dos 2 de la parte superior para formar un 4, mientras que se ha puesto otro 2 de forma aleatoria. Este movimiento incrementará la puntuación de la partida en 4 puntos o, lo que es lo mismo, el valor de las fichas nuevas obtenidas.

El juego terminará cuando, después de insertar una ficha aleatoria, no se puede realizar ningún movimiento. Un ejemplo de este fenómeno puede ser:

16	8	2	4
2	4	32	16
64	2	4	8
2	16	32	16

Ilustración 8 Ejemplo de derrota

2.3. Dificultad del juego

Aunque a primera vista, y debido a la sencillez de sus reglas, pueda parecer sencillo, no es para nada trivial. Se trata de un juego complejo debido al gran factor aleatorio que conlleva cada movimiento.

Cada vez que se realiza una acción, al tablero se le añade una nueva ficha. Si el jugador sólo se centra en emparejar fichas para conseguir una mayor, es relativamente sencillo aislar piezas de pequeño valor, lo que significa perder espacio, el cual es vital para poder continuar jugando. Por esto, este puzle se debe primar la liberación de espacio junto con la obtención de piezas más grandes.

Sin embargo, este doble objetivo hace difícil las decisiones sobre qué movimiento es mejor en un determinado momento. Muchos jugadores no se dan cuenta de este problema y continúan intentando conseguir piezas lo más grandes posibles.

En cuanto a los ordenadores, la dificultad también reside en la gran carga computacional y de cálculo que conlleva el uso de algoritmos de búsqueda, si bien la representación de la información de los tableros y la definición de las reglas o normas que definen como tomar movimientos es extremadamente sencilla.

Debido a estas dificultades, se propone usar varias técnicas de Inteligencia Artificial para intentar resolver el juego de la mejor forma posible.

2.4. Estudio de la complejidad del problema

En este apartado, se hará un estudio de la complejidad del problema.

En primer lugar, se realizará un estudio sobre las fichas conseguidas, la base de la puntuación. Suponiendo que para conseguir una ficha 4 se necesitan dos movimientos, uno para colocar dos fichas 2 y otro para juntarlas, se puede deducir que para conseguir una ficha 8 se necesitan 5 movimientos (el mismo proceso previo, pero repetido dos veces más un movimiento para juntar las fichas 4). Esta deducción es una aproximación, puesto que un mismo movimiento puede hacer las veces de varios o al revés. Sin embargo, ésta es una buena manera de comprender la complejidad del problema.

En la siguiente gráfica, se muestran los movimientos necesarios para conseguir las diferentes fichas de forma visual.

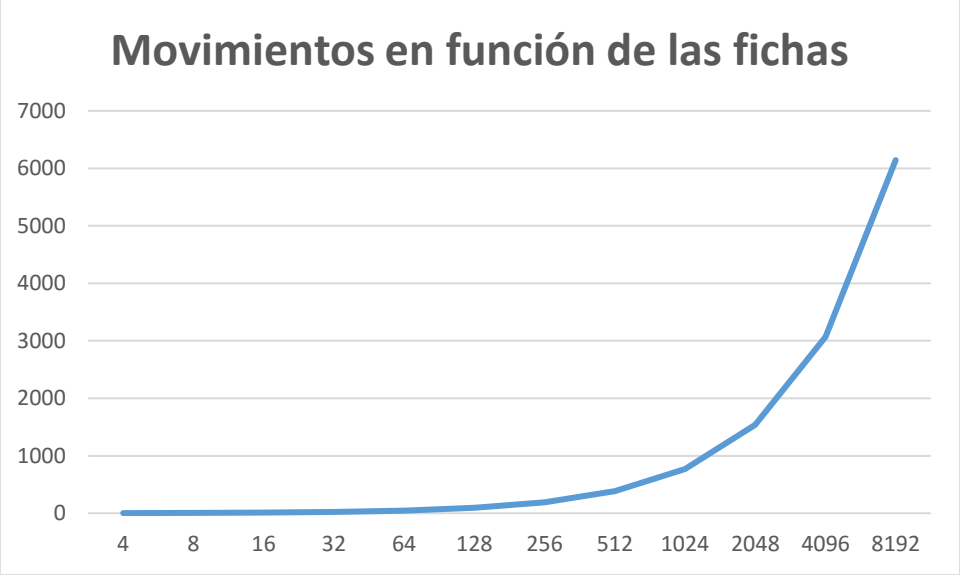


Ilustración 9 Movimientos en función de las fichas

Como se puede apreciar, queda una función exponencial, lo que significa que, a mayor ficha, mayor complejidad. Por lo tanto, se puede afirmar que este problema tiene una complejidad exponencial.

En cuanto a la clase de complejidad, debido a que se puede encontrar una solución al problema, se trata de un problema de tipo NP. En adicción, se ha demostrado que es un problema NP-Completo, como se muestra en los artículos [1], [2], [5] y [8], que se pueden encontrar en las referencias de este trabajo. Este tipo de problemas se caracterizan por poder resolverse con una máquina de Turing o de estados no determinista o, dicho con otras palabras, se puede encontrar un algoritmo que sea capaz de encontrar una solución al problema, sea cual sea el estado en el que se encuentre.

De forma adicional, es interesante comentar que el juego no es ilimitado. Debido al azar que interviene en cada movimiento, es inevitable que en algún momento el propio juego fuerce la pérdida. El mejor tablero que se puede conseguir en un modo de juego de 4x4 tendrá un aspecto parecido a este:

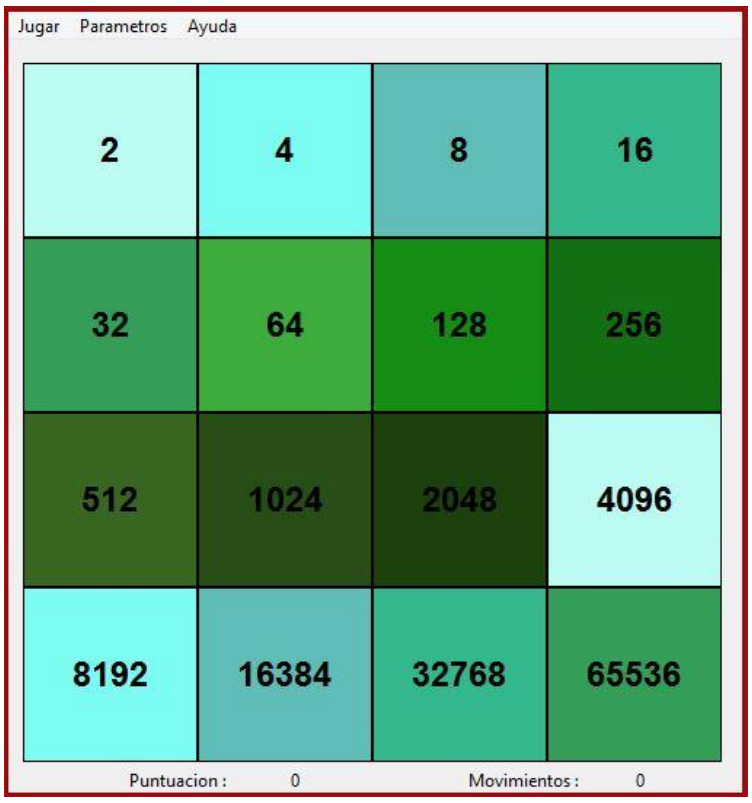


Ilustración 10 Posible tablero máximo

Como se puede apreciar, en cada casilla tiene una potencia de dos desde el propio 2 (2^1) hasta el 65536 (2^{16}). En cuanto se tuviera un tablero de este tipo, el azar llenará el único hueco libre, acabando el juego.

2.5. Algoritmos eficientes para la resolución del problema 2048

A continuación, se describirán una serie de algoritmos los cuales han demostrado ser eficientes en la resolución de este problema.

2.5.1. Algoritmos de búsqueda

Existen muchos algoritmos para la realización de este tipo de problemas. Un algoritmo de búsqueda consiste en una serie de pasos con los cuales podemos deducir las distintas posibilidades del juego, así como sus diversos beneficios o penalizaciones. Se basan en árboles de búsqueda, donde cada nodo representa un posible estado meta. Este árbol se va expandiendo en función de las reglas del juego hasta alcanzar un final.

2.5.1.1. Algoritmo "Minimax"

El algoritmo "*Minimax*" es un algoritmo recursivo que se utiliza en los juegos con oponente para minimizar la pérdida en el turno del rival, así como maximizar el beneficio en caso del turno del jugador. Se trata de un algoritmo de primero en profundidad, lo que significa que primero evalúa un nodo hasta la profundidad predeterminada y luego pasa a evaluar el resto.

Si bien es cierto que en el "2048" no hay oponente, se ha enfocado desde el punto de vista de que el jugador lucha contra el azar. Por lo tanto, a la hora de minimizar las pérdidas, lo que se intenta es predecir qué casilla aleatoria se escogerá para tenerla en cuenta en las decisiones de los movimientos.

Funciona de este modo:

- Se expande el primer nodo, que corresponderá a un movimiento del jugador. Se progresa hasta llegar a la profundidad escogida.
- Una vez allí, se evalúan los nodos hoja en función de la función de evaluación asignada y se va subiendo por el árbol. Dependiendo del turno que se trate:
    - Si es el turno del jugador que maximiza, se evaluarán todos los movimientos. Se tomará el mayor de los valores.
    - Si es el turno del jugador que minimiza, se evaluarán todas las posibilidades. Se tomará el menor de los valores.
- Una vez evaluados los movimientos, se elegirá como ganador al que tenga mayor puntuación.

A continuación, se muestra un ejemplo de ejecución del algoritmo *Minimax*:

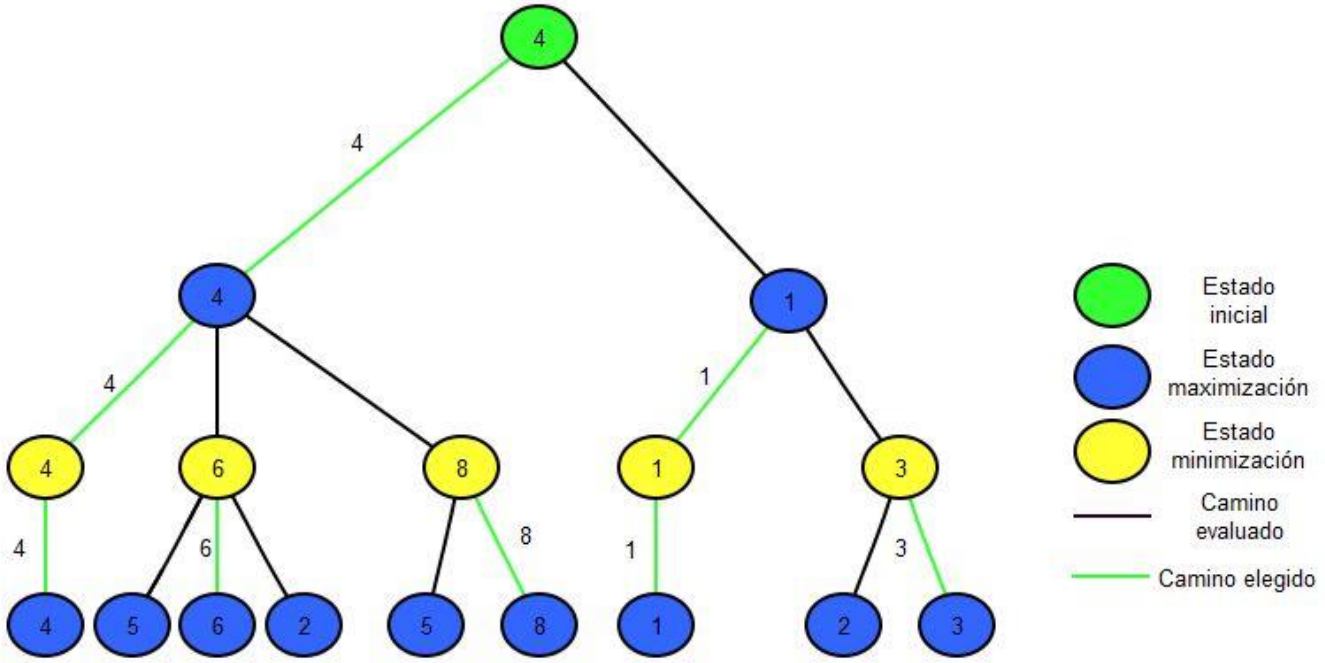


Ilustración 11 Algoritmo Minimax

Para más información, consultar la referencia [10] al final de esta memoria.

2.5.1.2. Algoritmo "α - β"

Existe una mejora del algoritmo *Minimax* llamado algoritmo "α - β", que sirve para evitar la evaluación de algunos nodos asegurando que el resultado final será el mismo que el devuelto por el *Minimax*. Es una gran mejora de tiempo, que permite ejecutar el algoritmo a profundidades mayores que el algoritmo *Minimax*.

La poda consiste en ir actualizando los valores de las dos variables α y β, siendo cada una la mejor y la peor opción de cada nodo. De forma inicial, estas variables tienen el valor menos infinito e infinito respectivamente. A continuación, se describe su funcionamiento:

- Se evalúa en profundidad el primer nodo hasta su hoja. Se devuelve el valor de este nodo.

- Se pasa a evaluar la siguiente rama del árbol:
  - Si el padre se trata de un paso de maximización, eso significa que el hijo es uno de minimización. El valor α será igual al mayor valor anteriormente evaluado, mientras que el β se irá actualizando. Si en algún momento β pasa a ser menor o igual que α, el resto de nodos pueden podarse, puesto que el valor resultante será siempre inferior a α. Por lo tanto, esta rama del árbol nunca será la ganadora y no es necesaria su evaluación.
  - En caso contrario, sucederá algo parecido. Se podarán todos los sucesores no explorados de cualquier nodo donde α sea mayor o igual que β.

A continuación, se muestra un ejemplo:

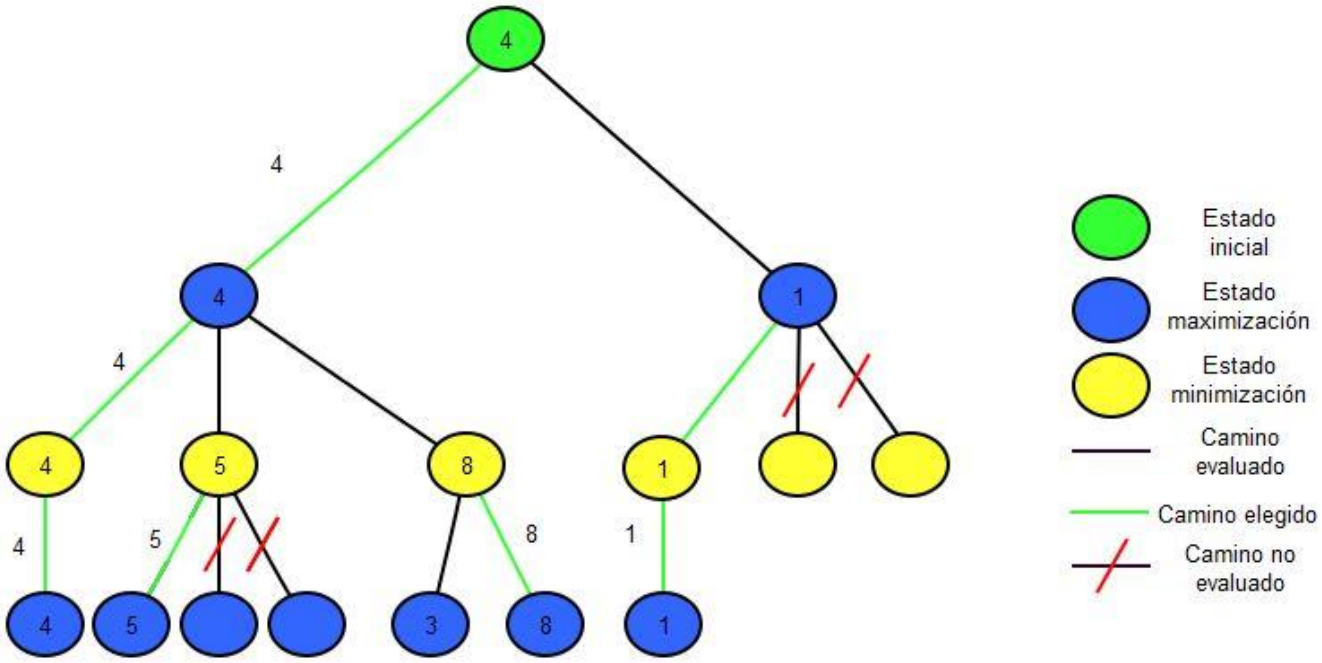


Ilustración 12 Ejemplo de poda "α-β"

Se va a proceder a explicar el anterior ejemplo para ayudar a comprender mejor el uso del algoritmo. Se comienza a evaluar el primer nodo y se baja hasta su hoja. Los valores de α y β son los iniciales puesto que aún no han sido modificados (-infinito e infinito respectivamente). El valor ganador es el único (4) por lo que ese valor subirá al siguiente nivel y actualizará el valor de α de esta comparación a 4. Además, el valor β del estado padre también se actualiza con este valor, ya que hasta ahora es la mejor opción.

Se procede a analizar la siguiente rama del árbol. En esta rama, el valor de α sigue siendo el inicial porque aún no se ha modificado el valor α del padre, mientras que β tendrá el valor 4. Se analiza el primer nodo, que tiene un valor 5. Se actualiza el valor α de este nodo a este valor. Al intentar analizar los siguientes nodos, salta la poda debido a que el valor de α es mayor que el de β. Esto significa que, al continuar analizando nodos, el valor ganador siempre será mayor que β y, por lo tanto, al intentar minimizar en el nodo padre nunca será ganador porque ya hay un valor inferior.

En el tercer nodo, el valor de α sigue siendo el inicial y el de β continuará siendo 4. Se analiza el primer nodo, obteniendo el valor 3. Se actualiza α con ese valor y, como α es menor que β, se siguen analizando nodos. El siguiente nodo tiene valor 8, por lo que el valor α se volverá a actualizar con este resultado y, al ser mayor que β, los siguientes nodos (si los hubiera) no serán analizados.

Se sube el valor de β hasta llegar a la raíz y se actualiza el valor α de la misma. Al analizar los siguientes nodos, el valor inicial de α será el de la raíz, en este caso 4. Se analiza la primera hoja de la siguiente rama, obteniendo un valor 1. Se actualiza el valor de β y, al volver a encontrarse con que α es mayor que β, todos los demás nodos no son analizados.

Este proceso se repite hasta que ya no queden nodos analizados. El ganador siempre será el mayor de los nodos hijo del estado inicial.

Para saber más sobre la poda α-β, consultar la referencia [10] contenida en la bibliografía de este proyecto.

2.5.2. Algoritmo de Monte Carlo

Otro método para resolver este tipo de juegos consiste en usar algoritmos de búsqueda no deterministas. Estos se basan en obtener estadísticas para la toma de decisión, por lo que con parámetros idénticos la decisión puede ser distinta.

Para la comprensión de este algoritmo se han utilizado las referencias [4], [7] y [11] que se pueden encontrar al final de este documento.

En este proyecto, se ha utilizado el Algoritmo de Monte Carlo. Este algoritmo ha sido utilizado con éxito para la toma de decisiones con juegos de dos jugadores como *Go*, haciendo uso tanto de exploración a altas profundidades como explotación de los resultados. Este algoritmo, concretamente el usado para el juego anteriormente mencionado, consta de cuatro fases:

1. Selección:
   Se selecciona el nodo del árbol que va a ser evaluado. Se va recorriendo el árbol desde la raíz hasta llegar al hijo que maximice la formula llamada UCT (*Upper Confidence bound applied to Trees*). Corresponderá con:

$$v_i + C \times \sqrt{\frac{\ln N}{n_i}}$$

   Donde v_i será el valor de las estadísticas obtenidas en cada hijo, *C* una constante que da valor relativo a la exploración, *N* será el número de llamadas totales del algoritmo y n_i el número de llamadas de cada hijo.

   Una vez llegado a un hijo donde todos sus posibles movimientos no han sido explorados, se pasará al siguiente paso.

2. Expansión:
   Se toma el hijo y se hace una jugada aleatoria sin explorar hasta ahora. Se tomará esta jugada como base para el siguiente paso.

3. Simulación:
   Se hacen una serie de jugadas aleatorias o pseudoaleatorias a partir del nodo base. Se recogen los resultados de cada una de ellas para obtener un resultado estadístico de todas ellas.

4. Propagación:
   Una vez obtenido el valor estadístico de la jugada en cuestión, se actualizan los valores de los antecesores correspondientes para la siguiente decisión.

Por medio de la repetición de este método, se puede llegar a tomar una decisión a una profundidad muy alta sin la necesidad del análisis de todos los nodos. Además, el algoritmo tiene la propiedad de “arrepentirse” de tomar un determinado nodo, puesto que si el resultado no es adecuado tomará otro camino en la siguiente iteración.

2.5.3. Optimización "Expectimax"

En la referencia de este trabajo, concretamente en la [9], se podrá encontrar el hasta ahora mejor controlador que utiliza inteligencia artificial para la resolución del juego 2048. Este programa utiliza la optimización "*Expectimax*", una mejora sobre los árboles de búsqueda creado por un algoritmo "*Minimax*".

Al igual que en este tipo de algoritmo, se toma el mejor movimiento en la fase de maximizar, pero en lugar de minimizar con hace el "*Minimax*", toma en consideración todas las posibilidades que tiene cada una de las casillas libres sin que sean evaluadas.

Esto ha obtenido unos resultados extraordinarios, no sólo alcanzando fichas de 2048 en cada partida, sino que alcanza piezas de valor 16384 en la mayoría de pruebas.

2.6. Resolución anterior del juego

El juego ha sido resuelto de muchas maneras diferentes. A continuación, se describirá brevemente algunas de ellas para tomarlas como punto de partida.

2.6.1. Humanos

Resulta complicado establecer una media global de resolución del juego 2048 por jugadores humanos debido a la falta de bases de datos al respecto. Sin embargo, existe una aplicación llamada *"2048 World Championship"* (enlace de descarga en la referencia [13]) que permite a las personas competir entre ellas para ver quién es el mejor jugador de 2048 del mundo.

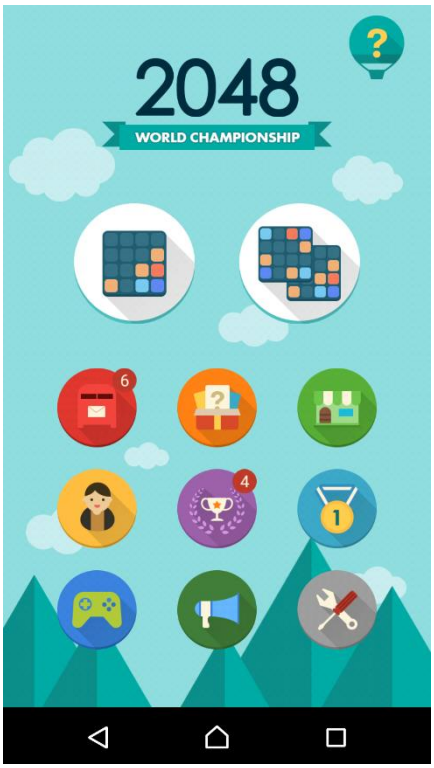


Ilustración 13 2048 World Championship

Pero la competición en sí no es importante para este proyecto. En cambio, existe una sección de puntuaciones que engloba todos los países que compiten, puntuando cada uno como la suma de las puntuaciones máximas de los jugadores de dicha nación.

Ilustración 14 Sección de puntuaciones por países del 2048 World Championship

Tomando estas puntuaciones como referencia, se han tomado la media de los mejores 10 países del mundo:

País	Puntuación total	Número jugadores	Puntuación media
Corea	349.862.499	12.630	27.700,91
Estados unidos	136.298.002	11.923	11.431,519
Japón	58.261.340	1.483	39.286,136
Brasil	53.622.401	4.732	11.331,868
Indonesia	45.698.293	1.573	29.051,68
Tailandia	43.909.534	1.748	25.119,871
Inglaterra	42.393.094	3.772	11.238,89
Taiwán	38.808.446	1.602	24.224,998
Rusia	38.133.621	3.582	10.645,902
India	37.668.149	2.147	17.544,55
TOTAL	**844.655.379**	**45.192**	**18.690,374**

Tabla 1 Puntuaciones máximas por países

Última comprobación de valores el día 21/05/2016.

Como se puede comprobar, según estos datos la media de puntuación máxima de los mejores jugadores humanos es de 18.690,374 puntos. Si bien no se cree que ésta sea la puntuación global que tiene el jugador humano en el juego, se considera una buena aproximación para este proyecto.

2.6.2. Concurso de controladores de IA

También se han realizado concursos para implementar controladores de Inteligencia Artificial que resuelvan este juego. Uno de ellos es el GECCO 2015 (referencia [12]), celebrado por la *Poznan University of Technology*. Las reglas de este concurso eran:

- Los controladores serán clasificados en función de la puntuación media
- Cada controlador deberá jugar al menos 1000 partidas para tener una puntuación estimada.
- Estar programado en Java
- El programa empleará 1 ms por acción tomada.

El ganador de este concurso fue un controlador que obtuvo una puntuación media de 18.245,6, con un porcentaje de acierto del 34,1%.

2.6.3. Controlador "*Expectimax*"

Hasta ahora, no se ha encontrado un controlador que funcione mejor que el descrito por la sección 2.5.3, página 35. Este programa ha batido todos los records, tanto humanos como de inteligencia artificial, obteniendo una media de puntuación aproximada de 390,000 puntos y alcanzando la ficha de 16.384 en su mayor parte de las partidas.

Esta puntuación es extremadamente alta, y no se pretende superar en este trabajo. Sin embargo, es interesante saber cuál es el mejor controlador hasta ahora para poder realizar comparativas más completas a lo largo del proyecto.

Capítulo 3
Objetivos

El objetivo final de este proyecto es crear un programa que sea capaz de elegir movimientos para jugar de la mejor forma posible en cualquier tablero de "2048". Utilizará algoritmos basados en Inteligencia Artificial y funciones de evaluación. A este programa se le podrá pasar cualquier tablero y podrá dar en un tiempo razonable el mejor movimiento para cada estado.

Los objetivos desglosados son:

1. **Análisis de un programa para la resolución del juego "2048":** Se hará un estudio exhaustivo del programa que se implementará y utilizará para la realización de este proyecto. Este estudio incluirá diseño de requisitos y diseño de casos de uso.

2. **Diseño de un programa para la resolución del problema "2048":** utilizando el análisis anterior, se procederá a diseñar el programa que resolverá el juego. Este diseño contendrá los diagramas de clases que cumplan los requisitos y casos de uso del objetivo 2.

3. **Implementación de un programa para la resolución del puzle "2048":** Se creará un programa que permitirá realizar pruebas del juego y consultar los resultados. Se podrá personalizar las configuraciones de las pruebas en función de distintos parámetros, como pueden ser tipo de tableros, profundidad, tipo de búsqueda y función de evaluación.

4. **Implementación de las reglas del juego:** el programa simulará los distintos movimientos del juego en función de las reglas previamente explicadas (Capítulo 2, sección 2.2, página 24).

5. **Implementación de una interfaz gráfica para ejecutar el puzle con distintas configuraciones:** Dicha interfaz permitirá recoger datos útiles para las pruebas posteriores, ya sea puntuación, tiempo tardado o movimientos ejecutados.

6. **Definición de funciones de evaluación**: Se definirán funciones de evaluación que intentarán resolver el juego siguiendo distintas estrategias.

7. **Implementación de algoritmos básicos para la resolución del puzle:** se implementará una serie de algoritmos muy simples que tratarán de resolver el problema mejor que un jugador puramente aleatorio.

8. **Implementación de algoritmos deterministas para la resolución del puzle:** utilizando la base teórica descrita en el capítulo anterior (capítulo 2, sección 2.5.1, página 30), se implementarán algoritmos deterministas que utilizarán las funciones de evaluación anteriores para resolver el juego.

9. **Implementación de algoritmos no deterministas para la resolución del puzle:** utilizando la base teórica descrita en el capítulo anterior (capítulo 2, sección 2.5.2, página 33), se implementarán algoritmos no deterministas que utilizarán las funciones de evaluación anteriores para resolver el juego.

10. **Resultados estadísticos:** Estudio exhaustivo de los resultados de las pruebas mediante la comparación entre ellas y los objetivos deseados. Las estadísticas obtenidas en cada prueba son la puntuación, el tiempo tardado y los movimientos realizados. Cada una de las pruebas corresponderá a una distinta configuración, entendiendo por configuración los distintos parámetros como el tipo de tablero, la función de evaluación utilizada, tipo de algoritmo, tipo de búsqueda y profundidad máxima.

11. **Resolución del juego estándar:** El programa tendrá que resolver el juego con el tablero clásico de 4x4 de la mejor forma posible. Para tener un punto de partida, se intentará obtener una puntuación superior a la media calculada en el punto 2.6.1 en la página 35 que, de manera redondeada, tiene el valor de 20.000 puntos.

12. **Conclusiones:** se realizará unas conclusiones exhaustivas para determinar si el proyecto es viable para la resolución de este puzle.

13. **Planificación de todo el proyecto:** se realizará una planificación de todo el proyecto, incluyendo presupuesto, planificación de tareas y planificación de recursos.

14. **Resumen en otro idioma:** en aras de hacer más accesible este proyecto, se realizará un resumen del mismo en inglés.

Capítulo 4
Desarrollo

Durante este capítulo, se describirá el proceso que ha seguido este proyecto. Se ha dividido en cinco partes:

- **Análisis del sistema:** Se estudiarán los requisitos y los casos de uso del proyecto.
- **Diseño del sistema:** Se realizará la arquitectura y los diagramas de clases del proyecto.
- **Implementación de la interfaz gráfica**: Se implementará una interfaz gráfica para que el programa pueda usarse de manera más sencilla.
- **Implementación de funciones de evaluación:** Se diseñará una serie de funciones de evaluación para resolver el problema.
- **Implementación de algoritmos:** Se tomarán los algoritmos base y se adecuarán para que encajen en este problema.

Todos estos pasos persiguen alcanzar los objetivos descritos en el capítulo 3, página 39, objetivos 1-9. El resto se irán alcanzando en los capítulos posteriores.

4.1. Análisis del sistema

Comenzando con el primer objetivo, se procederá a realizar un análisis del sistema. Este análisis consistirá en descomponer la tarea principal en subtareas más simples, para poder ir realizándolas de manera conjunta.

4.1.1. Descripción general del sistema

El programa a implementar será utilizado para realizar pruebas sobre el juego clásico de 2048 que permitirá tanto al usuario como al desarrollador sacar conclusiones sobre cuál es un buen método para resolver el puzle.

Esta herramienta estará dirigida a todos aquellos usuarios que quieran saber más sobre el problema del 2048. Además, el programa será muy fácilmente configurable con una serie de algoritmos y funciones de evaluación que se ejecutarán para resolver el juego.

4.1.2. Requisitos de usuario

A continuación, se procederá a especificar los requisitos de usuario. Estos requisitos son aquellos que se emplean para satisfacer las necesidades propias del usuario. Existen dos tipos:

- Requisito de capacidad: Define aquello que debe hacer el sistema y qué comportamiento tendrá ante distintas situaciones.
- Requisito de sistema: Define los servicios que usará y las restricciones que deberá cumplir el sistema.

Se utilizará la siguiente tabla para definir todos los requisitos:

Identificador			
Nombre			
Descripción			
Fuente		Prioridad	
Necesidad		Estabilidad	

Tabla 2 Modelo de tabla de requisitos

Cada parte corresponderá a:

- **Identificador:** identificador unitario que permitirá identificar con facilidad los requisitos.
- **Nombre:** breve descripción del requisito.
- **Descripción:** Explicación clara, ordenada y detallada del requisito.
- **Fuente:** indica el origen del requisito.
- **Necesidad:** Muestra el grado de obligatoriedad de los requisitos. Tomará los valores *alta*, *media* y *baja*.
- **Prioridad:** Indica la importancia de cada requisito. Tomará los valores *alta*, *media* y *baja*.
- **Estabilidad:** indica la frecuencia con la que un requisito puede variar. Tomará los valores *alta*, *media* y *baja*.

4.1.2.1. Requisitos de capacidad

Los requisitos de capacidad tendrán la nomenclatura basándose en el siguiente modelo:

RUC-XX

- R: Requisito.
- U: Usuario.
- C: Capacidad.
- XX: número del requisito.

Los requisitos de capacidad son:

Identificador	*Nombre*
RUC-01	Jugar manualmente
RUC-02	Jugar manual con parámetros introducidos
RUC-03	Jugar de forma automática
RUC-04	Cambiar tablero
RUC-05	Cambiar función de evaluación
RUC-06	Cambiar algoritmo de búsqueda
RUC-07	Cambiar profundidad
RUC-08	Mostrar ayuda
RUC-09	Salir del programa
RUC-10	Ver la puntuación
RUC-11	Ver los movimientos

Tabla 3 Listado de requisitos de capacidad

Identificador	RUC-01		
Nombre	Jugar manualmente.		
Descripción	El usuario podrá jugar de forma manual al juego de 2048.		
Fuente	Usuario	Prioridad	Baja
Necesidad	Media	Estabilidad	Alta

Tabla 4 Requisito de capacidad RUC-01

Identificador	RUC-02		
Nombre	Jugar manualmente con parámetros introducidos		
Descripción	El usuario tendrá un comando que permitirá al sistema seleccionar un movimiento en función de los parámetros introducidos (función de evaluación, algoritmo de búsqueda y profundidad.)		
Fuente	Usuario	Prioridad	Baja
Necesidad	Baja	Estabilidad	Alta

Tabla 5 Requisito de capacidad RUC-02

Identificador	RUC-03		
Nombre	Jugar de forma automática		
Descripción	El sistema podrá ejecutar partidas de forma automática utilizando los parámetros introducidos (función de evaluación, algoritmo de búsqueda y profundidad.)		
Fuente	Usuario	Prioridad	Alta
Necesidad	Alta	Estabilidad	Media

Tabla 6 Requisito de capacidad RUC-03

Identificador	RUC-04		
Nombre	Cambiar tablero		
Descripción	El usuario podrá seleccionar sobre qué tipo de tablero querrá ejecutar las pruebas.		
Fuente	Usuario	Prioridad	Media
Necesidad	Media	Estabilidad	Alta

Tabla 7 Requisito de capacidad RUC-04

Identificador	RUC-05		
Nombre	Cambiar función de evaluación		
Descripción	El usuario podrá seleccionar qué función de evaluación se utilizará para ejecutar las pruebas.		
Fuente	Usuario	Prioridad	Alta
Necesidad	Alta	Estabilidad	Alta

Tabla 8 Requisito de capacidad RUC-05

Identificador	RUC-06		
Nombre	Cambiar de algoritmo de búsqueda		
Descripción	El usuario podrá seleccionar qué algoritmo de búsqueda se utilizará para ejecutar las pruebas.		
Fuente	Usuario	Prioridad	Alta
Necesidad	Alta	Estabilidad	Alta

Tabla 9 Requisito de capacidad RUC-06

Identificador	RUC-07		
Nombre	Cambiar profundidad		
Descripción	El usuario podrá seleccionar qué profundidad máxima utilizará el algoritmo de búsqueda para ejecutar las pruebas.		
Fuente	Usuario	Prioridad	Alta
Necesidad	Alta	Estabilidad	Alta

Tabla 10 Requisito de capacidad RUC-07

Identificador	RUC-08		
Nombre	Mostrar ayuda		
Descripción	El programa mostrará el manual de usuario para ayudar al usuario a utilizar la herramienta.		
Fuente	Usuario	Prioridad	Media
Necesidad	Media	Estabilidad	Alta

Tabla 11 Requisito de capacidad RUC-08

Identificador	RUC-09		
Nombre	Salir del programa		
Descripción	El usuario podrá cerrar el programa en cualquier momento y situación.		
Fuente	Usuario	Prioridad	Baja
Necesidad	Baja	Estabilidad	Alta

Tabla 12 Requisito de capacidad RUC-09

Identificador	RUC-010		
Nombre	Ver puntuación		
Descripción	La puntuación obtenida por la partida manual actual se mostrará en la interfaz del programa.		
Fuente	Usuario	Prioridad	Baja
Necesidad	Baja	Estabilidad	Alta

Tabla 13 Requisito de capacidad RUC-10

Identificador	RUC-11		
Nombre	Ver movimientos		
Descripción	Los movimientos realizados por la partida manual actual se mostrarán en la interfaz del programa.		
Fuente	Usuario	Prioridad	Baja
Necesidad	Baja	Estabilidad	Alta

Tabla 14 Requisito de capacidad RUC-11

4.1.2.2. Requisitos de sistema

Los requisitos de sistema tendrán la nomenclatura basándose en el siguiente modelo:

RUS-XX

- R: Requisito.
- U: Usuario.
- S: Restricción.
- XX: número del requisito.

Los requisitos de sistema son:

Identificador	*Nombre*
RUS-01	Idioma
RUS-02	Lenguaje de programación
RUS-03	Formación de usuarios
RUS-04	Windows
RUS-05	Usabilidad

Tabla 15 Listado de requisitos de restricción

Identificador	RUS-01		
Nombre	Idioma		
Descripción	El idioma de la herramienta será el castellano		
Fuente	Usuario	Prioridad	Alta
Necesidad	Alta	Estabilidad	Alta

Tabla 16 Requisito de restricción RUS-01

Identificador	RUS-02		
Nombre	Lenguaje de programación		
Descripción	La herramienta estará programada en el lenguaje Python.		
Fuente	Usuario	Prioridad	Alta
Necesidad	Alta	Estabilidad	Alta

Tabla 17 Requisito de restricción RUS-02

Identificador	RUS-03		
Nombre	Formación de los usuarios		
Descripción	El programa contendrá suficiente información para que el usuario pueda utilizarlo sin problemas.		
Fuente	Usuario	Prioridad	Baja
Necesidad	Media	Estabilidad	Alta

Tabla 18 Requisito de restricción RUS-03

Identificador	RUS-04		
Nombre	Windows		
Descripción	El sistema se ejecutará de forma correcta en todas las versiones del sistema operativo Windows.		
Fuente	Usuario	Prioridad	Baja
Necesidad	Media	Estabilidad	Media

Tabla 19 Requisito de restricción RUS-04

Identificador	RUS-05		
Nombre	Usabilidad		
Descripción	El sistema tendrá unos criterios de usabilidad para facilitar su uso.		
Fuente	Usuario	Prioridad	Media
Necesidad	Media	Estabilidad	Media

Tabla 20 Requisito de restricción RUS-05

4.1.3. Requisitos de software

Estos serán los requisitos técnicos que utilizarán los requisitos del usuario para que el desarrollador del sistema pueda implementarlo sin problemas. Existen dos tipos:

- **Requisitos funcionales:** especifican lo que el programa deberá realizar.
- **Requisitos no funcionales:** especifican las restricciones en el sistema, así como el proceso de realización del mismo. Existen varios tipos dentro de este grupo:
    - **Requisitos de operación**: indican las acciones que el sistema realizará para el correcto funcionamiento del mismo.
    - **Requisitos de interfaz**: describe las características que deberá cumplir la interfaz.
    - **Requisitos de rendimiento**: restricciones de rendimiento que debe cumplir el sistema.
    - **Requisitos de recursos:** especificaciones de los recursos necesarios para la ejecución correcta del programa.
    - **Requisitos de usabilidad:** normas de usabilidad que permitirán al usuario utilizar el sistema de forma sencilla.
    - **Requisitos de comprobación:** requisitos que indican que la información procesada por el sistema tiene el formato correcto.

4.1.3.1. Requisitos funcionales

Los requisitos funcionales tendrán la nomenclatura basándose en el siguiente modelo:

RSF-XX

- R: Requisito.
- S: Software.
- F: Funcionales.
- XX: número del requisito.

Identificador	Nombre
RSF-01	Realizar movimiento arriba
RSF-02	Realizar movimiento abajo
RSF-03	Realizar movimiento izquierda
RSF-04	Realizar movimiento derecha
RSF-05	Realizar movimiento en función de los parámetros
RSF-06	Realizar una partida automática
RSF-07	Realizar pruebas
RSF-08	Cambiar tablero
RSF-09	Cambiar función de evaluación
RSF-10	Cambiar algoritmo
RSF-11	Cambiar profundidad
RSF-12	Mostrar ayuda

Tabla 21 Listado de requisitos funcionales

Identificador	RSF-01		
Nombre	Realizar movimiento arriba		
Descripción	El sistema podrá ejecutar un movimiento hacia arriba, tanto en el modo manual como en el automático.		
Fuente	Analista	Prioridad	Alta
Necesidad	Alta	Estabilidad	Alta

Tabla 22 Requisito funcional RSF-01

Identificador	RSF-02		
Nombre	Realizar movimiento abajo		
Descripción	El sistema podrá ejecutar un movimiento hacia abajo, tanto en el modo manual como en el automático.		
Fuente	Analista	Prioridad	Alta
Necesidad	Alta	Estabilidad	Alta

Tabla 23 Requisito funcional RSF-02

Identificador	RSF-03		
Nombre	Realizar movimiento izquierda		
Descripción	El sistema podrá ejecutar un movimiento hacia la izquierda, tanto en el modo manual como en el automático.		
Fuente	Analista	Prioridad	Alta
Necesidad	Alta	Estabilidad	Alta

Tabla 24 Requisito funcional RSF-03

Identificador	RSF-04		
Nombre	Realizar movimiento derecha		
Descripción	El sistema podrá ejecutar un movimiento hacia la derecha, tanto en el modo manual como en el automático.		
Fuente	Analista	Prioridad	Alta
Necesidad	Alta	Estabilidad	Alta

Tabla 25 Requisito funcional RSF-04

Identificador	RSF-05		
Nombre	Realizar un movimiento en función de los parámetros		
Descripción	El sistema podrá ejecutar un movimiento utilizando la función de evaluación, el algoritmo de búsqueda y la profundidad introducidas por el usuario, tanto en el modo manual como en el automático.		
Fuente	Analista	Prioridad	Alta
Necesidad	Alta	Estabilidad	Alta

Tabla 26 Requisito funcional RSF-05

Identificador	RSF-06		
Nombre	Realizar una partida de forma automática		
Descripción	El sistema ejecutará movimientos de forma automática utilizando los parámetros especificados por el usuario (función de evaluación, algoritmo de búsqueda y profundidad) hasta que el juego se detenga.		
Fuente	Analista	Prioridad	Alta
Necesidad	Alta	Estabilidad	Alta

Tabla 27 Requisito funcional RSF-06

Identificador	RSF-07		
Nombre	Realizar pruebas de forma automática		
Descripción	El sistema deberá ejecutar un determinado número de partidas automáticas.		
Fuente	Analista	Prioridad	Alta
Necesidad	Alta	Estabilidad	Alta

Tabla 28 Requisito funcional RSF-07

Identificador	RSF-08		
Nombre	Cambiar tablero		
Descripción	El sistema deberá cambiar el tablero de juego cada vez que el usuario lo desee.		
Fuente	Analista	Prioridad	Alta
Necesidad	Alta	Estabilidad	Alta

Tabla 29 Requisito funcional RSF-08

Identificador	RSF-09		
Nombre	Cambiar función de evaluación		
Descripción	El sistema deberá cambiar la función de evaluación a utilizar cada vez que el usuario lo desee.		
Fuente	Analista	Prioridad	Alta
Necesidad	Alta	Estabilidad	Alta

Tabla 30 Requisito funcional RSF-09

Identificador	RSF-10		
Nombre	Cambiar algoritmo de búsqueda		
Descripción	El sistema deberá cambiar el algoritmo de búsqueda a utilizar cada vez que el usuario lo desee.		
Fuente	Analista	Prioridad	Alta
Necesidad	Alta	Estabilidad	Alta

Tabla 31 Requisito funcional RSF-10

Identificador	RSF-11		
Nombre	Cambiar profundidad		
Descripción	El sistema deberá cambiar la profundidad utilizada por el algoritmo de búsqueda cada vez que el usuario lo desee.		
Fuente	Analista	Prioridad	Alta
Necesidad	Alta	Estabilidad	Alta

Tabla 32 Requisito funcional RSF-11

Identificador	RSF-12		
Nombre	Mostrar ayuda		
Descripción	El sistema mostrará el manual de usuario del programa cada vez que el usuario lo desee.		
Fuente	Analista	Prioridad	Baja
Necesidad	Media	Estabilidad	Alta

Tabla 33 Requisito funcional RSF-12

4.1.3.2. Requisitos no funcionales

Los requisitos no funcionales tendrán la nomenclatura basándose en el siguiente modelo:

RSNF-XX

- R: Requisito.
- S: Software.
- NF: No funcionales.
- XX: número del requisito.

Identificador	Nombre
RSNF-01	Windows
RSNF-02	Escritura en ficheros
RSNF-03	Interfaz clara
RSNF-04	Interfaz sencilla
RSNF-05	Idioma
RSNF-06	Movimientos rápidos
RSNF-07	Crear fichero
RSNF-08	Parámetros por defecto
RSNF-09	Lenguaje de programación
RSNF-10	Manual de usuario
RSNF-11	Estructura de ficheros

Tabla 34 Listado de requisitos no funcionales

4.1.3.2.1. Requisitos no funcionales de operación

Identificador	RSNF-01		
Nombre	Windows		
Descripción	El sistema deberá ser compatible con el sistema operativo Windows en todas sus versiones.		
Fuente	Analista	Prioridad	Media
Necesidad	Media	Estabilidad	Media

Tabla 35 Requisito no funcional RSNF-01

Identificador	RSNF-02		
Nombre	Escritura en fichero		
Descripción	Al finalizar una partida, el sistema escribirá los resultados y otra información de interés en un fichero con un nombre representativo.		
Fuente	Analista	Prioridad	Media
Necesidad	Alta	Estabilidad	Alta

Tabla 36 Requisito no funcional RSNF-02

4.1.3.2.2. Requisitos no funcionales de interfaz

Identificador	RSNF-03		
Nombre	Interfaz clara		
Descripción	El sistema tendrá una interfaz clara y amigable para el usuario.		
Fuente	Analista	Prioridad	Baja
Necesidad	Media	Estabilidad	Alta

Tabla 37 Requisito no funcional RSNF-03

Identificador	RSNF-04		
Nombre	Interfaz sencilla		
Descripción	El sistema tendrá una interfaz sencilla e intuitiva para el usuario.		
Fuente	Analista	Prioridad	Baja
Necesidad	Media	Estabilidad	Alta

Tabla 38 Requisito no funcional RSNF-04

Identificador	RSNF-05		
Nombre	Idioma		
Descripción	El idioma de la aplicación resultante será el castellano.		
Fuente	Analista	Prioridad	Baja
Necesidad	Media	Estabilidad	Alta

Tabla 39 Requisito no funcional RSNF-05

4.1.3.2.3. Requisitos no funcionales de rendimiento

Identificador	RSNF-06		
Nombre	Movimientos rápidos		
Descripción	Los movimientos realizados por el sistema no deberán tardar más de 0,005 segundos en ser ejecutados.		
Fuente	Analista	Prioridad	Media
Necesidad	Media	Estabilidad	Alta

Tabla 40 Requisito no funcional RSNF-06

4.1.3.2.4. Requisitos no funcionales de recursos

Identificador	RSNF-07		
Nombre	Crear fichero		
Descripción	El sistema creará un fichero con un nombre identificativo para escribir los resultados de las pruebas en caso de no existir ya.		
Fuente	Analista	Prioridad	Baja
Necesidad	Media	Estabilidad	Alta

Tabla 41 Requisito no funcional RSNF-07

Identificador	RSNF-08		
Nombre	Parámetros por defecto		
Descripción	El sistema tendrá una serie de parámetros por defecto para la ejecución de las pruebas. Estos parámetros corresponderán a la ejecución de una prueba aleatoria.		
Fuente	Analista	Prioridad	Media
Necesidad	Media	Estabilidad	Alta

Tabla 42 Requisito no funcional RSNF-08

Identificador	RSNF-09		
Nombre	Lenguaje de programación		
Descripción	El lenguaje de programación utilizado será Python.		
Fuente	Analista	Prioridad	Alta
Necesidad	Alta	Estabilidad	Alta

Tabla 43 Requisito no funcional RSNF-09

4.1.3.2.5. Requisitos no funcionales de usabilidad

Identificador	RSNF-10		
Nombre	Manual de usuario		
Descripción	El sistema pondrá a disposición del usuario un manual de instrucciones para ayudarle en el uso del programa.		
Fuente	Analista	Prioridad	Baja
Necesidad	Media	Estabilidad	Alta

Tabla 44 Requisito no funcional RSNF-10

4.1.3.2.6. Requisitos no funcionales de comprobación

Identificador	RSNF-11		
Nombre	Estructura de ficheros		
Descripción	El sistema utilizará siempre el mismo formato para la escritura de ficheros para facilitar su lectura.		
Fuente	Analista	Prioridad	Baja
Necesidad	Media	Estabilidad	Alta

Tabla 45 Requisito no funcional RSNF-11

4.1.4. Casos de uso

Los casos de uso son los diferentes escenarios a los que nuestro programa se va a enfrentar. A continuación, se procederá a detallarlos para aclarar la funcionalidad del programa a diseñar.

En este caso, el único actor que actúa sobre el programa será el propio usuario, y los casos de uso del mismo serán:

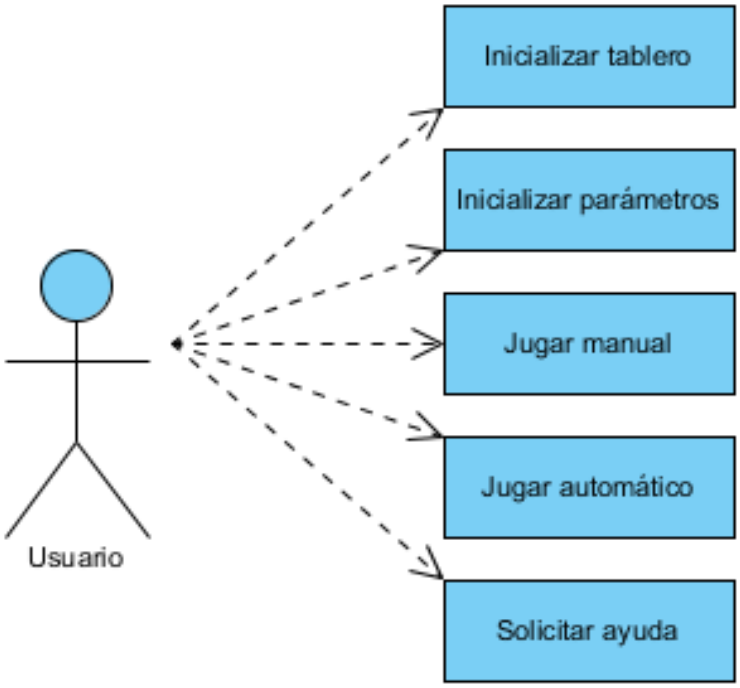


Ilustración 15 Casos de uso

Los casos de uso tendrán los siguientes apartados:

- **Código:** Identificador unitario para poder hacer referencia a un único caso de uso.
- **Nombre:** Nombre del caso de uso.
- **Descripción:** Descripción del caso de uso.
- **Actores:** Actores que influyen en el caso de uso.
- **Precondiciones:** Condiciones previas que se deben cumplir para ejecutar el caso de uso.
- **Postcondiciones:** Condiciones posteriores que se deben cumplir al ejecutar el caso de uso.
- **Pasos:** Secuencia que se irá ejecutando a medida que el caso de uso transcurra.

CU-001

Nombre	Inicializar tablero.
Descripción	Crear un tablero nuevo con ciertos parámetros (algoritmo, profundidad, función de evaluación).
Actores	Usuario.
Precondiciones	Abrir el programa.
Postcondiciones	-
Pasos	1- El usuario selecciona en opción del menú "*Parámetros*". 2- El usuario selecciona en opción del menú anterior "*Tableros*". 3- El usuario selecciona el tablero que desea utilizar.

Tabla 46 Caso de Uso CU-001

CU-002

Nombre	Inicializar parámetros.
Descripción	Seleccionar los parámetros para las pruebas.
Actores	Usuario.
Precondiciones	Abrir el programa.
Postcondiciones	-
Pasos	1- El usuario selecciona en opción del menú "*Parámetros*". 2- El usuario selecciona en opción del primer menú "*Función de evaluación*". 3- El usuario selecciona la función de evaluación que desea utilizar. 4- El usuario selecciona en opción del primer menú "*Algoritmo*". 5- El usuario selecciona el algoritmo que desea utilizar. 6- El usuario selecciona en opción del primer menú "*Profundidad*". 7- El usuario selecciona la profundidad que desea utilizar.

Tabla 47 Caso de Uso CU-002

CU-003

Nombre	Jugar en modo manual.
Descripción	Jugar de manera manual, o bien seleccionando los movimientos o bien dejando que el algoritmo seleccionado los determine.
Actores	Usuario.
Precondiciones	Seleccionar los parámetros y tablero deseados.
Postcondiciones	-
Pasos	1- El usuario selecciona en opción del menú "*Jugar*". 2- El usuario selecciona en opción del menú anterior "*Jugar manual*". 3- El tablero se actualizará para empezar la partida. A partir de aquí, el usuario puede utilizar las teclas predeterminadas para jugar.

Tabla 48 Caso de Uso CU-003

CU-004

Nombre	Jugar con parámetros.
Descripción	El programa ejecuta un gran número de partidas de manera automática utilizando los parámetros utilizados anteriormente (algoritmo, profundidad, función de evaluación).
Actores	Usuario.
Precondiciones	Seleccionar los parámetros y tablero deseados.
Postcondiciones	-
Pasos	1- El usuario selecciona en opción del menú "*Jugar*". 2- El usuario selecciona en opción del menú anterior "*Jugar Algoritmo*". 3-Se comenzará una nueva partida hasta llegar al número máximo de partidas. 4-Cuando el tablero se llene y no quede ningún movimiento posible, guardará las estadísticas de la partida y pasará a 3.

Tabla 49 Caso de Uso CU-004

CU-005

Nombre	Solicitar ayuda
Descripción	Revisar el uso del programa
Actores	Usuario.
Precondiciones	Abrir el programa.
Postcondiciones	-
Pasos	1- El usuario selecciona en opción del menú "*Ayuda*". 2- El usuario recibirá un enlace donde podrá descargar un pdf que contenga el manual de usuario.

Tabla 50 Caso de Uso CU-005

4.1.5. Matrices de trazabilidad

Requisitos de capacidad	Casos de uso				
	CU-01	CU-02	CU-03	CU-04	CU-05
RUC-01			X		
RUC-02			X		
RUC-03				X	
RUC-04	X				
RUC-05		X			
RUC-06		X			
RUC-07		X			
RUC-08					X
RUC-09	X	X	X	X	X
RUC-10			X		
RUC-11			X		

Tabla 51 Matriz de trazabilidad entre casos de uso y requisitos de capacidad

Requisitos de restricción	Casos de uso				
	CU-01	CU-02	CU-03	CU-04	CU-05
RUR-01	X	X	X	X	X
RUR-02	X	X	X	X	X
RUR-03					X
RUR-04	X	X	X	X	X
RUR-05	X	X	X	X	X

Tabla 52 Matriz de trazabilidad entre casos de uso y requisitos de restricción

Requisitos de capacidad	Requisitos funcionales											
	RSF-01	RSF-02	RSF-03	RSF-04	RSF-05	RSF-06	RSF-07	RSF-08	RSF-09	RSF-10	RSF-11	RSF-12
RUC-01	X	X	X	X								
RUC-02					X							
RUC-03						X	X					
RUC-04								X				
RUC-05									X			
RUC-06										X		
RUC-07											X	
RUC-08												X
RUC-09												
RUC-10	X	X	X	X	X							
RUC-11	X	X	X	X	X							

Tabla 53 Matriz de trazabilidad entre requisitos funcionales y requisitos de capacidad

Requisitos de restricción	Requisitos funcionales											
	RSF-01	RSF-02	RSF-03	RSF-04	RSF-05	RSF-06	RSF-07	RSF-08	RSF-09	RSF-10	RSF-11	RSF-12
RUR-01												
RUR-02												
RUR-03												
RUR-04												
RUR-05												

Tabla 54 Matriz de trazabilidad entre requisitos funcionales y requisitos de restricción

Requisitos de capacidad	Requisitos no funcionales										
	RSNF-01	RSNF-02	RSNF-03	RSNF-04	RSNF-05	RSNF-06	RSNF-07	RSNF-08	RSNF-09	RSNF-10	RSNF-11
RUC-01	X		X	X	X	X	X	X	X		X
RUC-02	X		X	X	X	X	X	X	X		X
RUC-03	X	X	X	X	X	X	X	X	X		X
RUC-04	X		X	X	X				X		
RUC-05	X		X	X	X				X		
RUC-06	X		X	X	X				X		
RUC-07	X		X	X	X				X		
RUC-08	X		X	X	X				X	X	
RUC-09	X		X	X	X				X		
RUC-10	X		X	X	X				X		
RUC-11	X		X	X	X				X		

Tabla 55 Matriz de trazabilidad entre requisitos no funcionales y requisitos de capacidad

Requisitos de restricción	Requisitos no funcionales										
	RSNF-01	RSNF-02	RSNF-03	RSNF-04	RSNF-05	RSNF-06	RSNF-07	RSNF-08	RSNF-09	RSNF-10	RSNF-11
RUR-01					X						
RUR-02									X		
RUR-03										X	
RUR-04	X										
RUR-05			X	X							

Tabla 56 Matriz de trazabilidad entre requisitos no funcionales y requisitos de restricción

4.2. Diseño del sistema

A continuación, se continuará con el segundo objetivo. Se mostrarán los diagramas de clases UML que definirán el diseño y estructura del programa.

Como lenguaje de programación, se ha decidido emplear *Python* debido a la sencillez de sus sentencias, además del gran material disponible en materia de creación de interfaces gráficas. En este caso, se ha decidido utilizar la librería *Wxpython* para la realización de la misma.

En este proyecto, se han definido cuatro clases:

- **Tablero:** Objeto que instanciará cada tablero que se pueda dar a lo largo de la ejecución del programa.
- **Partida:** Definición de las funciones que determinarán las reglas del juego, así como alguna otra auxiliar de apoyo necesarias para la ejecución de determinados algoritmos.
- **Interfaz:** Interfaz de apoyo al usuario para utilizar el programa.
- **Algoritmo:** Conjunto de funciones que tomarán decisiones sobre cuál será el mejor movimiento en función del algoritmo determinado por el usuario.

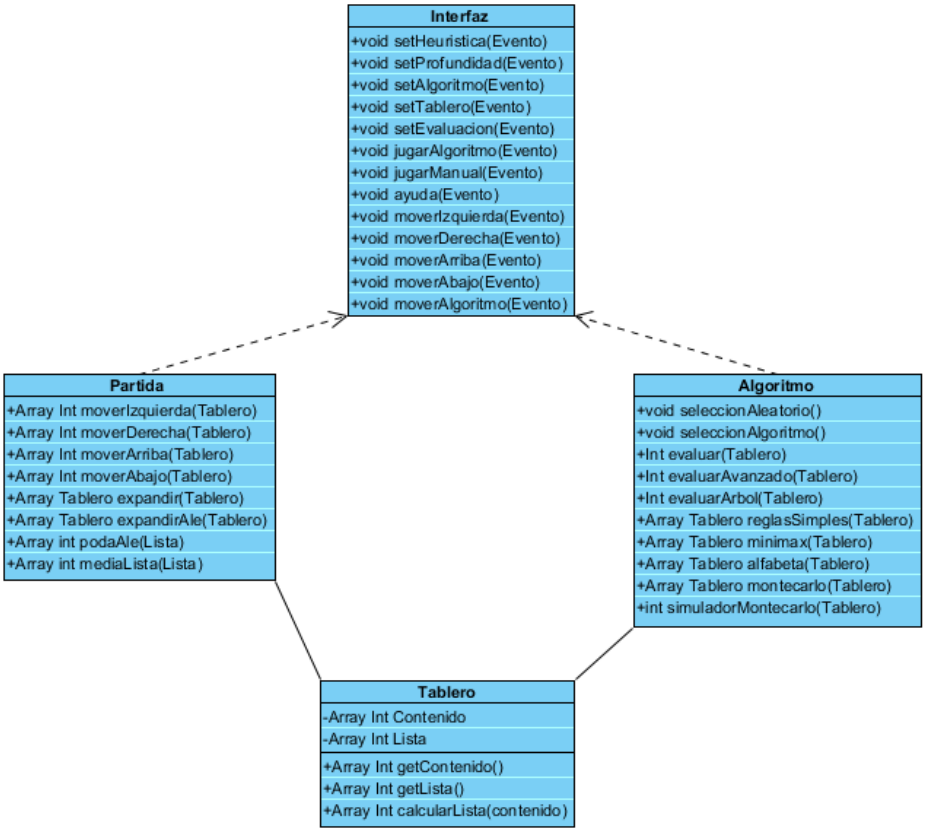


Ilustración 16 Diagrama de Clases

4.2.1. Tablero

Esta clase es la base de todo el proyecto. Con ella, se podrá crear instancias de tableros para la ejecución de la mayoría de las funciones del programa. Un tablero contendrá un contenido, es decir, el valor que tiene cada casilla, y una lista que indica las posiciones libres del mismo. Este último atributo resultará muy útil para determinados algoritmos de búsqueda.

Como operadores, estarán los *get* de los atributos, así como una función que calcule la lista asociada a un tablero.

Ilustración 17 Diagrama Tablero

4.2.2. Partida

Clase que aportará la funcionalidad de las normas básicas del juego "2048". Los operadores de movimiento realizarán el determinado movimiento en función del tablero que se le suministra por parámetro, para a continuación añadir una ficha nueva en una casilla vacía aleatoria. Gracias a estas funciones, se podrá ejecutar las reglas del juego tal y como se pretendía en el objetivo número 4.

De forma adicional, tenemos las funciones:

- ***expandir(Tablero)*:** Devolverá una lista de tableros correspondientes a los posibles movimientos del asignado por parámetros.
- ***expandirAle(Tablero)*:** Devolverá una lista de tableros cada uno con su probabilidad del azar.
- ***podaAle(Lista)*:** Método que reducirá las posibilidades del azar a una lista de valores representativos.
- ***mediaLista(Lista)*:** Método de apoyo para darle peso a cada valor de la poda anterior en función de los nodos que representa. Utilizada con el algoritmo "*Minimax*" para realizar la media en su fase de minimización.

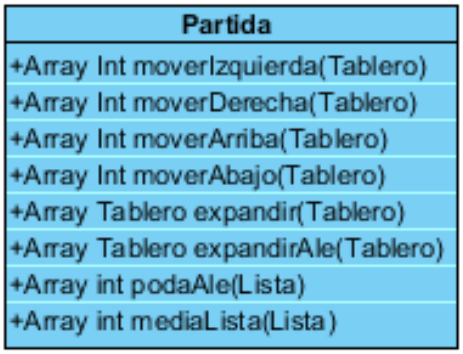


Ilustración 18 Diagrama Partida

4.2.3. Interfaz

Esta clase implementa de forma gráfica la configuración del programa para que al usuario le sea más sencillo de usar. En ella, se podrá alterar los distintos parámetros necesarios para la ejecución de los algoritmos mediante eventos producidos por la interacción del usuario con el menú de la aplicación.

Existen eventos especiales que no se producirán con la ya mencionada interacción del usuario con el menú. Estos eventos son los que provocan los movimientos en el caso de que el usuario deseara jugar de forma manual.

Para más información sobre el uso de la aplicación, consulte el manual de usuario contenido en los anexos del presente documento.

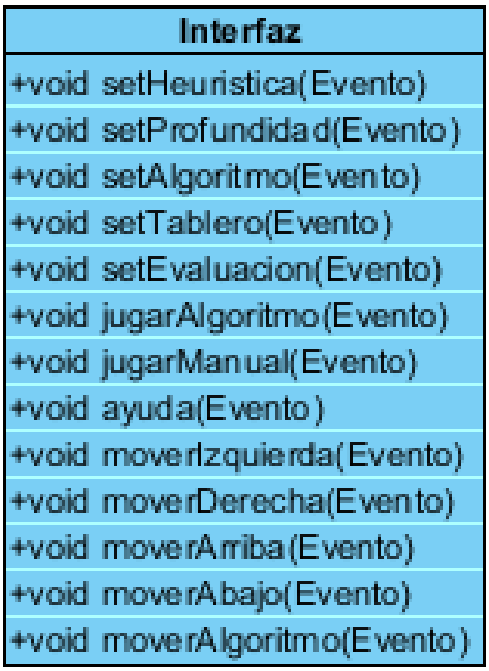


Ilustración 19 Diagrama Interfaz

4.2.4. Algoritmo

Los algoritmos que resuelven el juego están contenidos en esta clase. Todos ellos reciben un tablero como parámetro y seleccionan el movimiento que creen óptimo según sus criterios. Además, en esta clase están las funciones *seleccionAleatorio()* y *seleccionAlgoritmo()*, que son las encargadas de llamar de manera recurrente al algoritmo de selección en el caso de que el usuario desee que la aplicación juegue de forma automática.

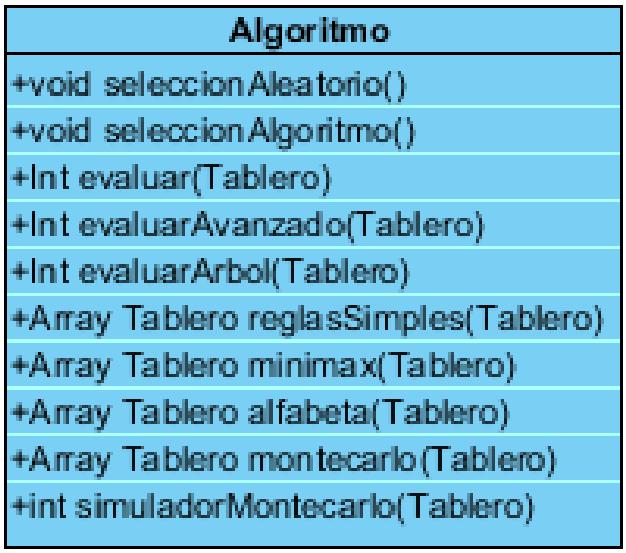


Ilustración 20 Diagrama Algoritmo

Siguiendo estos diagramas de clases y al análisis explicado en la página 43, se podrá implementar el programa perseguido por el objetivo número 3.

4.3. Interfaz

El objetivo número 5 consiste en crear una interfaz gráfica lo más amigable posible que permita tanto al diseñador como al usuario utilizar el programa diseñado anteriormente. Se decidió emplear la librería *Wxpython* para realizar una interfaz muy simple e intuitiva. Es la siguiente:

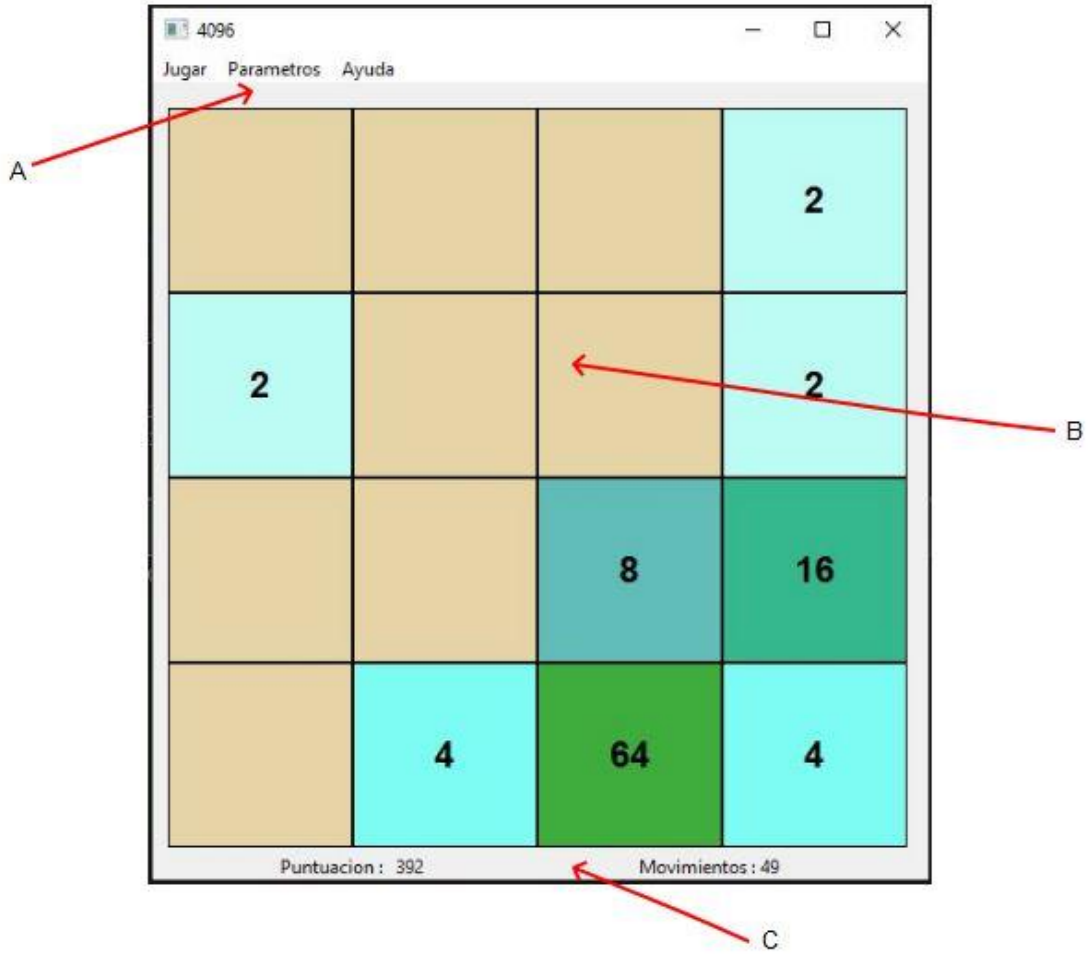


Ilustración 21 Interfaz gráfica implementada

Donde:

- **A:** Menús desplegables donde el usuario y diseñador podrá interactuar con el programa.

- **B:** Tablero actual. Se decidió darle la mayor importancia al mismo, ocupando casi la totalidad de la pantalla. Como también se puede apreciar, se han utilizado números grandes y en negrita para facilitar al usuario su lectura. De forma adicional, se ha utilizado un degradado de color que ayudará al usuario relacionar los números que se pueden juntar debido a la similitud de color.

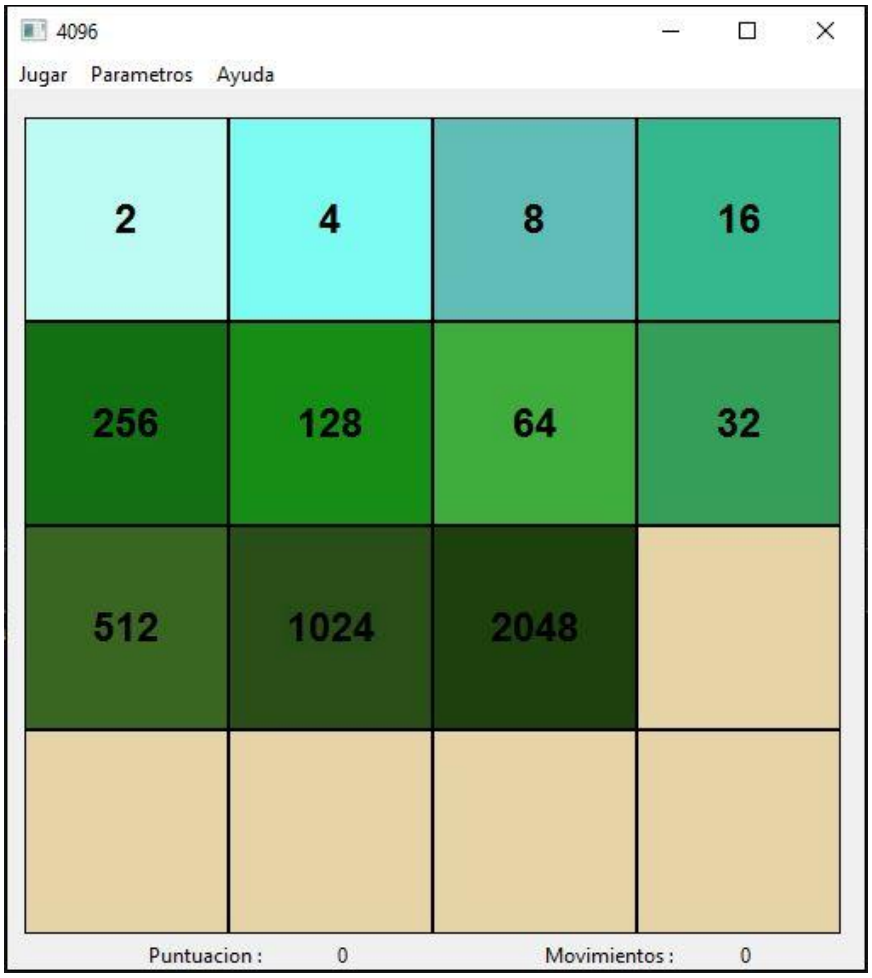


Ilustración 22 Degradado de color de la interfaz

- **C:** Resumen de la puntuación y los movimientos realizados en la partida actual.

En caso de duda sobre cómo utilizar esta interfaz, se podrá consultar el manual de usuario en el Anexo I, página 134.

4.4. Funciones de evaluación

Como se vio en el objetivo 6 del capítulo 3, será necesaria la definición de unas funciones de evaluación que permitan al programa resolver el problema siguiendo distintas estrategias. Una función de evaluación es una función computable que proporciona información sobre la disposición de, en este caso, los tableros de "2048". Se han definido y probado las siguientes funciones en la evaluación de los distintos árboles de búsqueda:

- "**Serpiente":** Su nombre proviene por su semejanza a una serpiente. El valor se calcula de la siguiente manera:

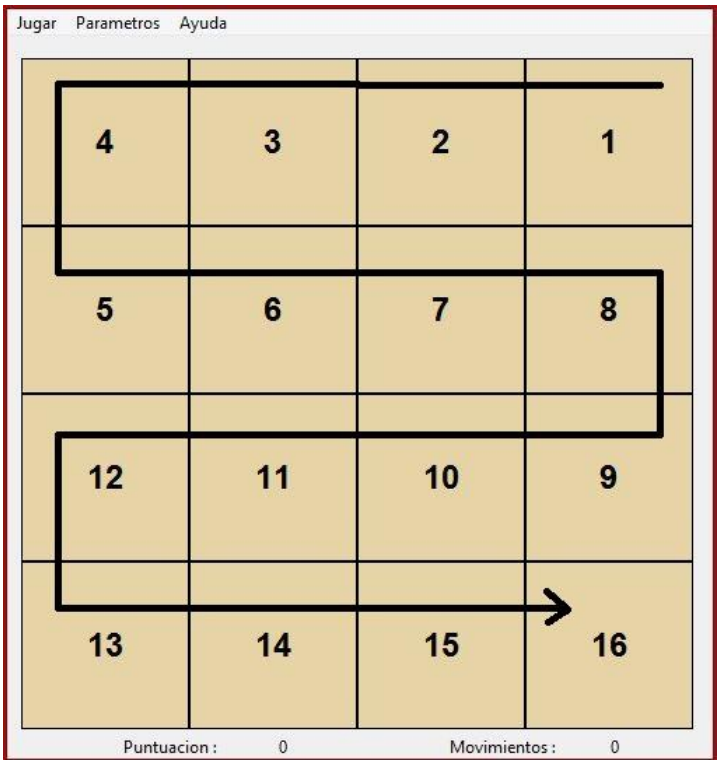


Ilustración 23 Función de evaluación serpiente

$$Ev = \sum casilla^{peso}$$

Dónde el peso es el valor de cada *casilla* y casilla el valor de la ficha que hay en la propia casilla.

Estos pesos se han elegido para que las fichas tiendan a ordenarse según la flecha de la ilustración. Por lo tanto, a medida que las piezas de valor consecutivo se vayan juntando, será mucho más sencillo obtener una de un valor superior. Además, se eligieron unos pesos consecutivos para promover que las fichas vayan ordenándose de mejor manera.

La idea de esta función de evaluación es darle importancia a la ordenación de las fichas. De este modo, las fichas pequeñas no se aislarán tanto como con un jugador puramente aleatorio. De forma adicional, la ordenación de una función de evaluación a otra no se realiza de la misma forma, por lo que se podrá comprobar cuál de ellas es mejor.

- **"Esquina":** En este caso, se ha sustituido la tabla de pesos para darle más importancia a las cuatro esquinas. Será de esta forma.

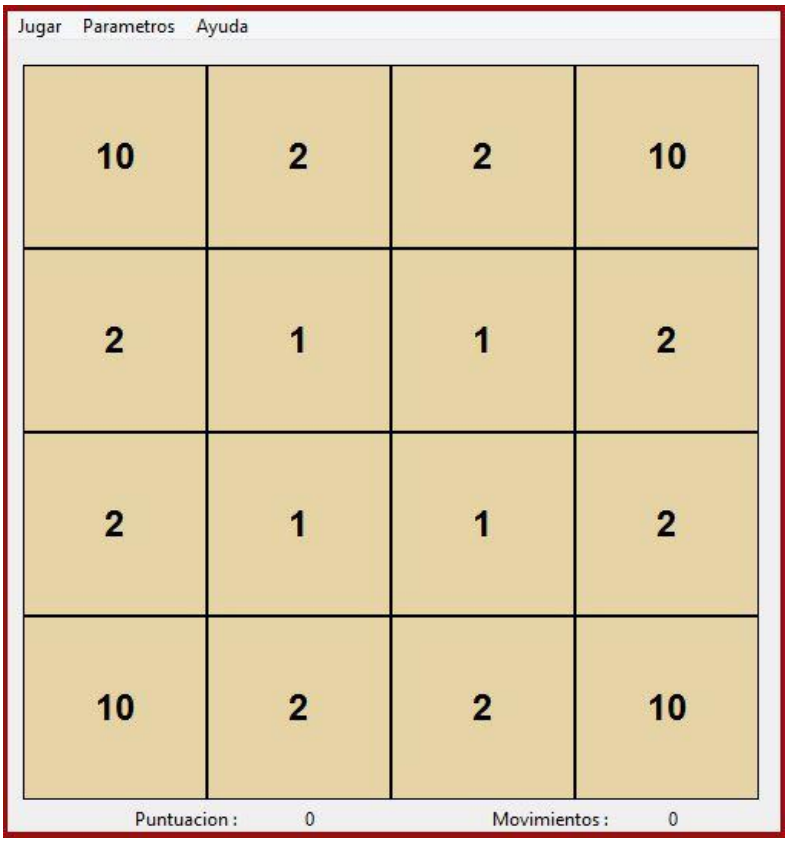


Ilustración 24 Función de evaluación esquina

El valor se obtendrá de igual forma que en el anterior caso:

$$Ev = \sum casilla^{peso}$$

Dónde el peso es el valor de cada *casilla* y casilla el valor de la ficha que hay en la propia casilla.

Estos pesos se han elegido para que el programa priorice las esquinas o las casillas inmediatamente adyacentes como posición para las fichas más grandes. De esta forma, se deberá ir acumulando en una determinada esquina y así será más fácil juntarlas.

- **"Puntuación"**: En este caso, el valor corresponderá únicamente con la puntuación acumulada.

Como se puede apreciar, la información transmitida por las funciones de evaluación va disminuyendo, siendo la "serpiente" la más informada y la "puntuación" la menos.

4.5. Diseño de los algoritmos básicos

El siguiente objetivo a alcanzar es el número 6, consistente en obtener una serie de algoritmos sencillos que sean capaces de resolver el problema mejor que un agente que elija movimientos puramente aleatorios.

Estos algoritmos se basan en explotar las funciones de evaluación anteriores, ya sea por medio de evaluación simple o con un pequeño árbol de búsqueda. También se usó un conjunto de reglas simples para poder comparar y determinar si el uso de árboles de búsqueda es una buena forma para resolver el problema.

4.5.1. Evaluación simple:

Este algoritmo toma un tablero y lo evalúa. Se elegirá el movimiento con el mayor valor, siguiendo estos pasos:

- Se toma un tablero.
- Se expanden sus sucesores.
- Se evalúan los propios sucesores gracias a una determinada función de evaluación (se eligió serpiente, debido a que era la más informada).
- Se selecciona el mayor como ganador

4.5.2. Evaluación avanzada con reglas simples:

Se tomará un tablero como las veces anteriores. A la hora de evaluar, se considerará mejor las fichas que tengan una debajo exactamente igual. Esto favorece que, a medida que pasa el juego, las fichas iguales se vayan agrupando para posteriormente juntarse y obtener una mayor. Esto se conseguirá simplemente multiplicando por 2 estas fichas. Por lo tanto, este algoritmo consistirá en:

- Coger cada ficha del tablero.
- Compararla con la que tiene debajo.
- Si no son iguales, evaluar de igual manera que en el algoritmo anterior.
- Si son iguales, multiplicar por dos la ficha y evaluar.
- Sumar todas las evaluaciones para obtener el valor total.

Por ejemplo, se tiene:

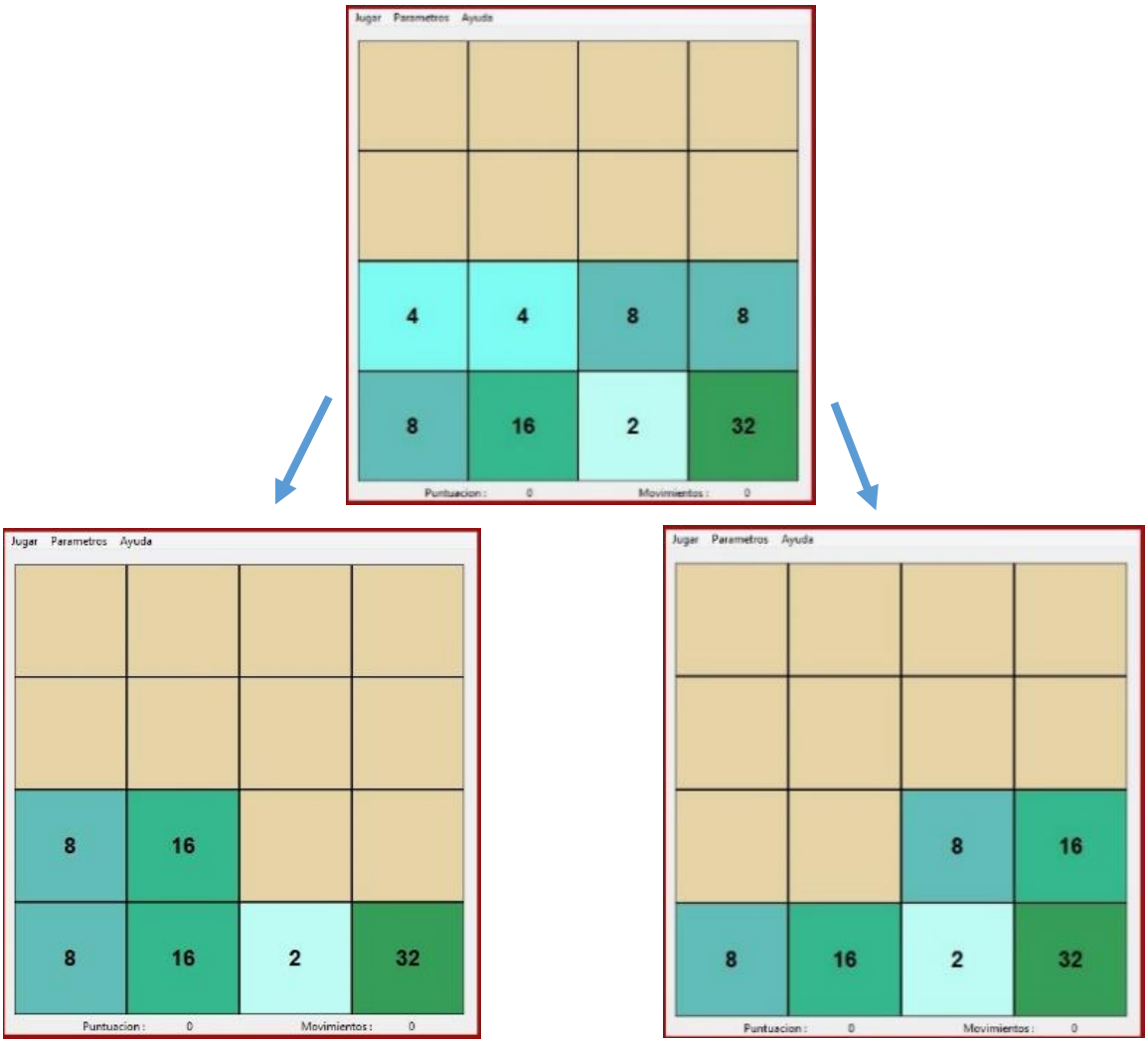


Ilustración 25 Evaluación por reglas

A pesar de que los sucesores tienen la misma puntuación, el de la izquierda conseguirá ordenar las fichas para conseguir una pieza mayor en unos pocos pasos. Con el método anterior, se priorizará la evaluación a corto plazo, mientras que con éste se elegirá el sucesor de la izquierda, con la esperanza de obtener mejores resultados.

4.5.3. Evaluación con árbol.

Otra forma de conseguir sucesores a medio plazo consiste en tomar un tablero y su lista para poder ver cuál de sus sucesores es el mejor. Se seguirán estos pasos que, como se puede deducir, se asemeja al funcionamiento de un árbol de búsqueda con profundidad dos:

- Se evalúa el tablero tantas veces como movimientos haya.
- Se expande el tablero.
- Se suma la evaluación de cada sucesor con la del padre.
- Se devuelve la mejor puntuación obtenida.

Sin embargo, este algoritmo no tiene en cuenta el azar, factor de vital importancia en este juego. Esta falta de información puede y provocará que muchas evaluaciones no obtenga el resultado esperado. Por lo tanto, a partir de ahora este factor aleatorio se tendrá muy en cuenta a la hora de tomar las decisiones.

4.6. Implementación de algoritmos deterministas

El siguiente objetivo, el número 8, consiste en implementar una serie de algoritmos deterministas adaptados a la resolución del problema del "2048". Un algoritmo determinista es aquel que, bajo un mismo escenario, obtiene el mismo resultado.

A la hora de diseñar estos algoritmos, todos ellos basados en los explicados en el apartado 2.5.1 del capítulo 2, página 30, se realizaron distintas estrategias o tipos de búsqueda. Estos tipos se basaban en la importancia que se le atribuía al azar que, como se vio en el apartado 4.5.3, tiene una importancia capital en este problema.

4.6.1. Búsqueda realista

Como se ha explicado en el capítulo 2, apartado 2.5.1.1, página 30, el algoritmo "*Minimax*" pretende maximizar las ganancias del jugador humano y minimizar las ganancias del azar. Sin embargo, este tipo de búsqueda es demasiado cautelosa, lo que puede ocasionar que no se consigan buenas puntuaciones debido a la falta de riesgos. Por lo tanto, se decidió modificar ligeramente este algoritmo para realizar lo que se llamará "búsqueda con riesgo" o "búsqueda realista".

Como ya se ha comentado, el turno que se minimizará es el asignado al azar. Al minimizar la pérdida en este sentido se realiza una búsqueda muy pesimista, puesto que siempre se da por hecho que el azar pondrá una ficha en el peor lugar posible. No obstante, eso no es cierto la mayoría de las veces, puesto que la ficha será asignada a una posición aleatoria. Teniendo en cuenta esta cuestión, se procedió a realizar una modificación del algoritmo "*Minimax*" para que asumiera más riesgos con la esperanza de alcanzar mejores puntuaciones jugando contra un oponente que, siendo aleatorio, no es en absoluto racional. Este cambio se trata simplemente de variar la minimización del turno del azar por realizar una media, como se ve en la siguiente imagen:

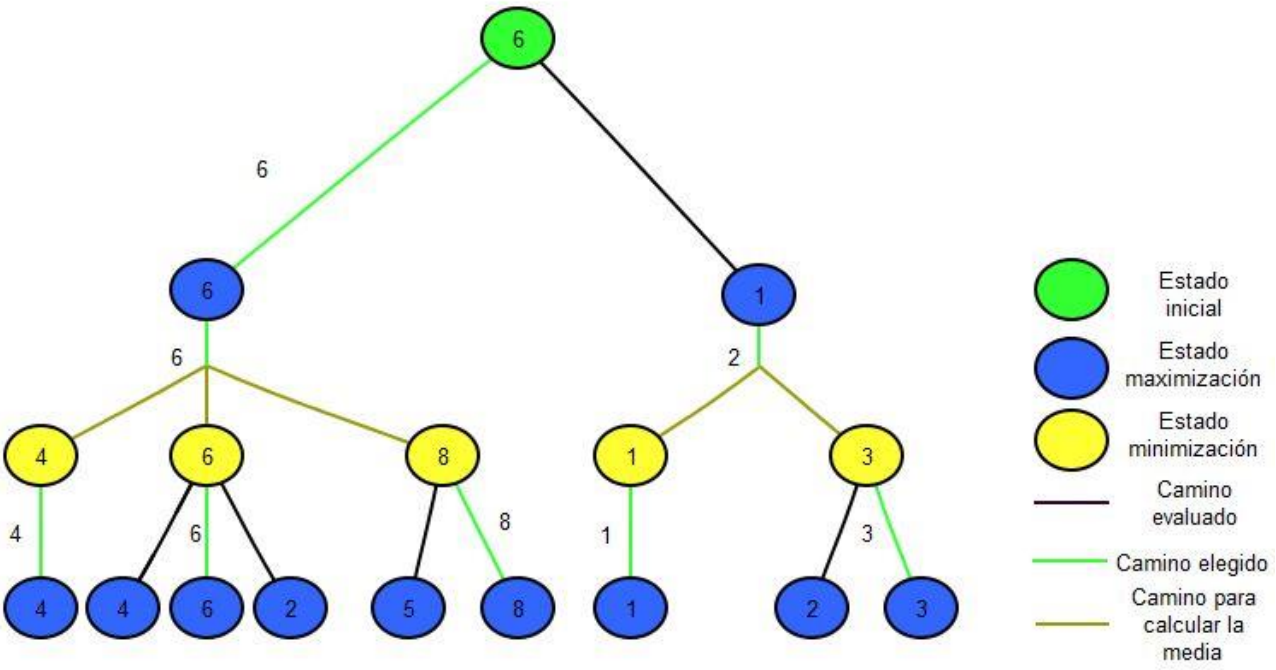


Ilustración 26 Algoritmo Minimax modificado

Como se puede apreciar en la imagen, en los estados de minimización ya no se toma el más pequeño. En su lugar, se cogen todos los estados y se halla una media para obtener el valor que corresponde con el padre.

Además, se han realizado algunas optimizaciones del juego para adaptarlo al uso de estos árboles de búsqueda.

El primero de estos cambios es el resumen de los nodos generados por el azar. Después de una observación detallada de estos nodos, se descubrió que había simetrías que se pueden explotar para agilizar la evaluación. Por ejemplo:

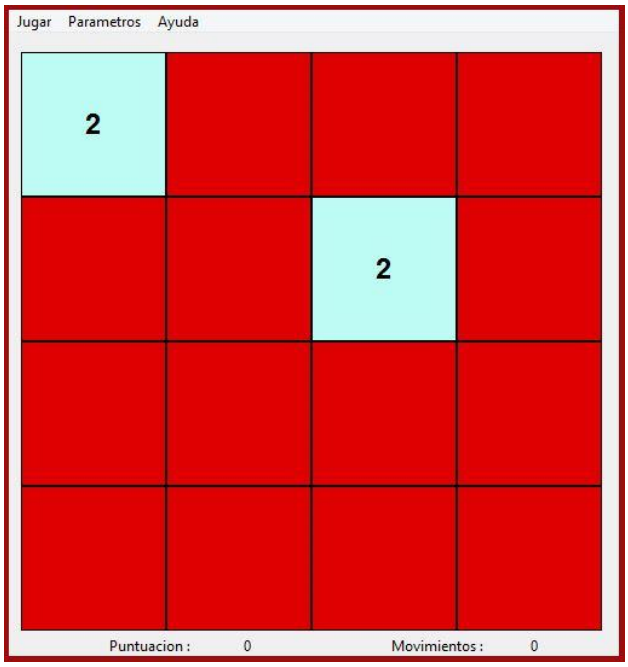


Ilustración 27 Resumen del azar

Las casillas de color rojo son todas las casillas donde el azar puede introducir fichas nuevas. En el siguiente movimiento del jugador, se pueden desplazar a cualquiera de las direcciones indicadas. Como se puede apreciar, en algunos casos un mismo movimiento sobre dos fichas producidas por el azar consigue el mismo tablero.

Suponiendo que al tablero anterior se le introduce una ficha aleatoria y luego se desplaza hacia la izquierda:

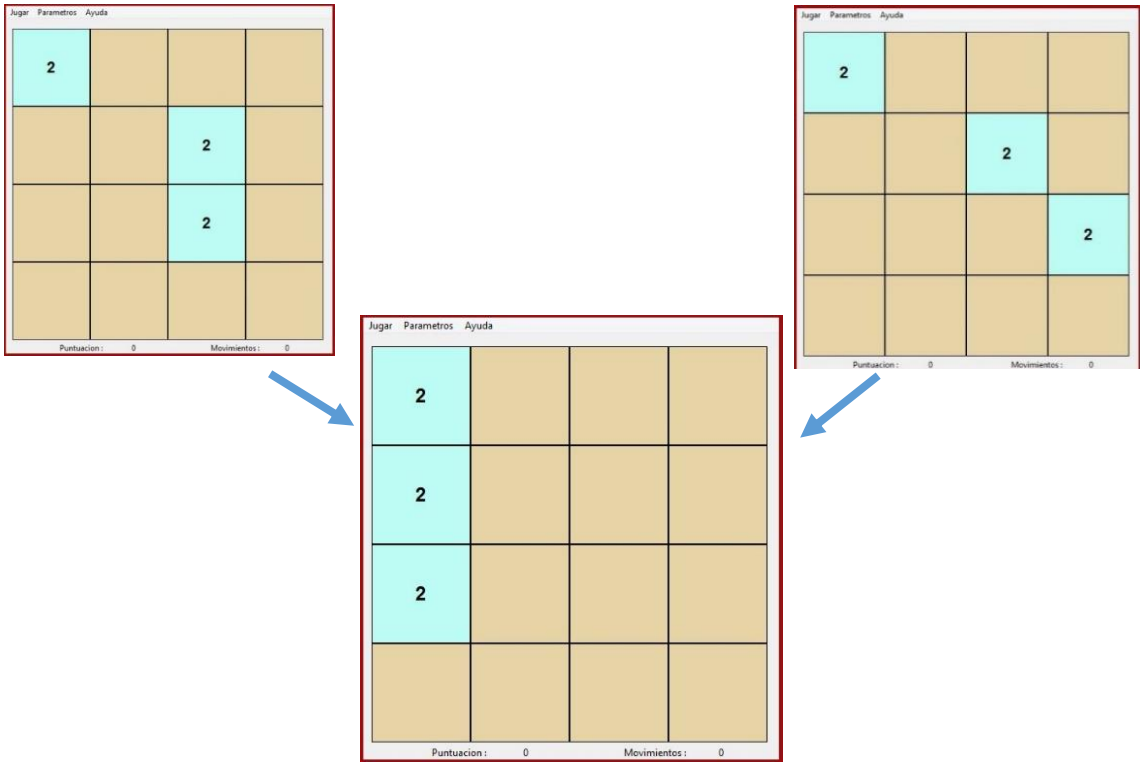


Ilustración 28 Posible ejemplo de azar

En el ejemplo anterior, se puede ver que si el jugador hace un movimiento izquierda, en ambos casos de azar se da el mismo tablero.

Por lo tanto, se creó un algoritmo que resume las fichas producidas eliminando la redundancia. El funcionamiento de este algoritmo es el siguiente:

- Se introducen en una lista las casillas vacías que rodean las que están llenas.
- Se buscan en esa lista los ángulos rectos de los posibles "triángulos" y se eliminan los ángulos rectos.
- Se introducen en la misma lista los bordes de las filas o columnas totalmente vacías, así como las esquinas.

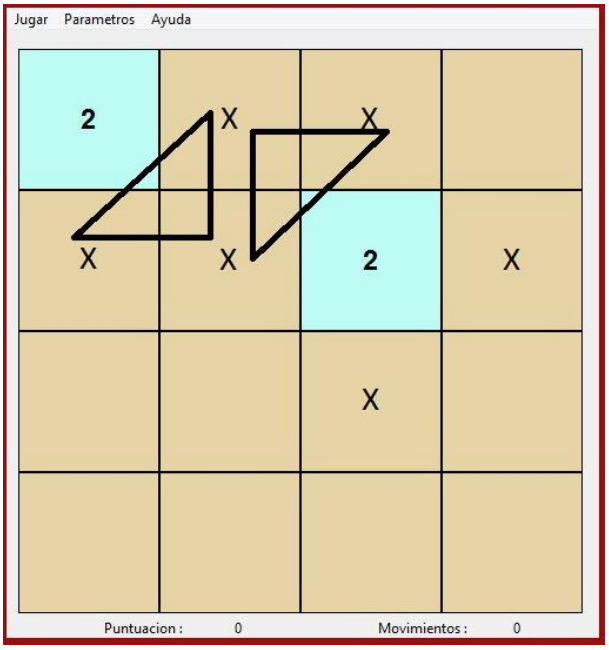


Ilustración 29 Ejemplos de "triángulo"

De esta forma, se eliminan aproximadamente un tercio de las posibilidades del azar, lo que producirá una mejora de tiempo exponencial a medida que aumenta la profundidad.

Debido a este cambio, ya no es posible hallar el mismo resultado de media que se obtendría de la evaluación de todos los nodos. Por lo tanto, fue necesario hacer una aproximación de esta media.

El método para hacerla se realiza a la vez que el resumen, añadiendo un peso a cada nodo resultante. El peso corresponderá al número de nodos que representa la simplificación.

4.6.2. Búsqueda pesimista

Otra posibilidad de búsqueda es ignorar todos los riesgos que se han tomado anteriormente y dar por hecho que la casilla más importante a la hora de determinar el azar es la que más perjudica al jugador humano. Con esto, se realiza una búsqueda mucho más segura, puesto que cada decisión tomada solamente tiene en cuenta el peor de los casos. Es posible que esto provoque una bajada de puntuación, pero al no haber riesgos se espera que las partidas duren más tiempo y, por lo tanto, tengan mejores resultados.

Este tipo de búsqueda se puede obtener con un algoritmo *Minimax* simple. Sin embargo, y debido a que el algoritmo "α - β" reduce en gran medida el número de nodos evaluados sin que el resultado se vea alterado, se decidió utilizar esta mejora. Con ella, se pretende obtener una mejora en tiempo de ejecución asegurando que el resultado no se verá afectado.

Por tanto, también se implementó el algoritmo α - β basándose en el uso del *Minimax* implementado anteriormente. Si bien no es necesario el cálculo de medias comentado en el apartado anterior, puesto que las simetrías no se toman en cuenta a la hora de calcular el descendiente en el turno de minimización. Para recordar el uso y funcionamiento de este algoritmo, se puede consultar el capítulo 2, apartado 5.2, página 26.

Sin embargo, para evitar al lector la búsqueda de dicha explicación, se mostrará a continuación el ejemplo utilizado en la sección mencionada anteriormente:

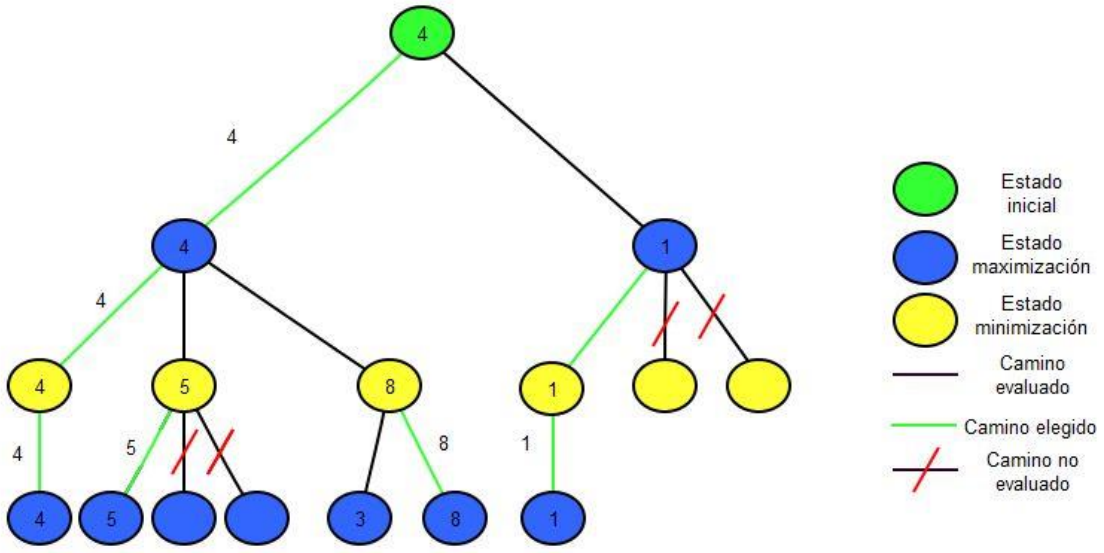


Ilustración 30 Ejemplo del uso del algoritmo α-β

4.6.3. Búsqueda optimista

La última posibilidad de búsqueda consiste en tomar en cuenta la mejor posibilidad del azar. El algoritmo para realizar esto sería un equivalente al "*Minimax*", pero cambiando la parte de minimización por otra de maximización.

La implementación de esta búsqueda sería muy sencilla, pero no se ha realizado debido a la teoría que la alta cantidad de riesgos tomada no favorecerá al árbol de búsqueda. A pesar de que esta tesis no ha sido probada, se espera que los resultados de los dos anteriores tipos de búsqueda determinen si puede ser o no acertada.

4.7. Algoritmos no deterministas

Continuando con los algoritmos utilizados para resolver el problema y siguiendo también el objetivo número 9, se procederá a realizar un algoritmo no determinista basado en el algoritmo de Monte Carlo explicado en el capítulo 2, sección 2.5.2, página 33.

Un algoritmo no determinista es aquel que, bajo un mismo escenario, dará lugar a resultados diferentes. Esto se produce debido a que no toma sus decisiones teniendo en cuenta todas las posibilidades, sino que lo hace tomando un subconjunto de las mismas y lo explota a grandes profundidades.

Este algoritmo de Monte Carlo ha sido modificado de forma importante para adecuarlo al problema del "2048", aunque se ha utilizado como base para el que se describirá a continuación.

El primer cambio importante es la fórmula con la que se calcula la ganancia de la explotación y la exploración. En lugar de usar UCT (véase Capítulo 2, Sección Algoritmo de Monte Carlo, página 24), se ha utilizado la siguiente:

$$x = valor(x) + C\,\frac{\sum valor(pruebas)}{nº\_pruebas}$$

La primera parte corresponderá a la exploración, y será simplemente la puntuación obtenida con respecto al tablero base debida a este movimiento. La segunda parte se tratará de la exploración, con una media de los valores obtenidos en este método multiplicados por la constante de valoración a la exploración.

El algoritmo funciona del siguiente modo:

- Se toma el tablero que se desea averiguar su mejor movimiento y se expanden sus posibles sucesores.

Ilustración 31 Expansión Monte Carlo

- A continuación, se realizan una serie de simulaciones con cada uno de los tableros resultantes. Se trata de jugadas sin compromiso de la siguiente forma:
    - Se coloca una ficha aleatoria.
    - Se selecciona un movimiento con algún criterio predeterminado. Estos criterios determinan la importancia que se les da a los posibles sucesores, dando la posibilidad de que, aunque uno de ellos parezca no tener buen resultado, pueda ser elegido por si acaso uno de sus sucesores si lo pueda obtener. Estos criterios pueden ser:
        - **Probabilístico:** a cada sucesor se le asigna una probabilidad de ser asignado basado en la puntuación obtenida.
        - **Directo:** el sucesor con mayor valor será seleccionado.
        - **Aleatorio:** el sucesor se elegirá de forma aleatoria

- Se continúa con este proceso hasta alcanzar la profundidad máxima o cuando el jugador pierda. Se devuelve los valores obtenidos de esta manera:

$$ValorTotal+=\frac{ValorProfundidadActual}{profundidadActual}$$

De forma que se obtendrán mayores valores si se consigue mucha puntuación a bajas profundidades. Esto dará prioridad a jugadas que consigan cambios importantes de la puntuación rápidamente, en lugar de muchas puntuaciones pequeñas. En caso de que se haya encontrado con una derrota, penalizará severamente este valor.

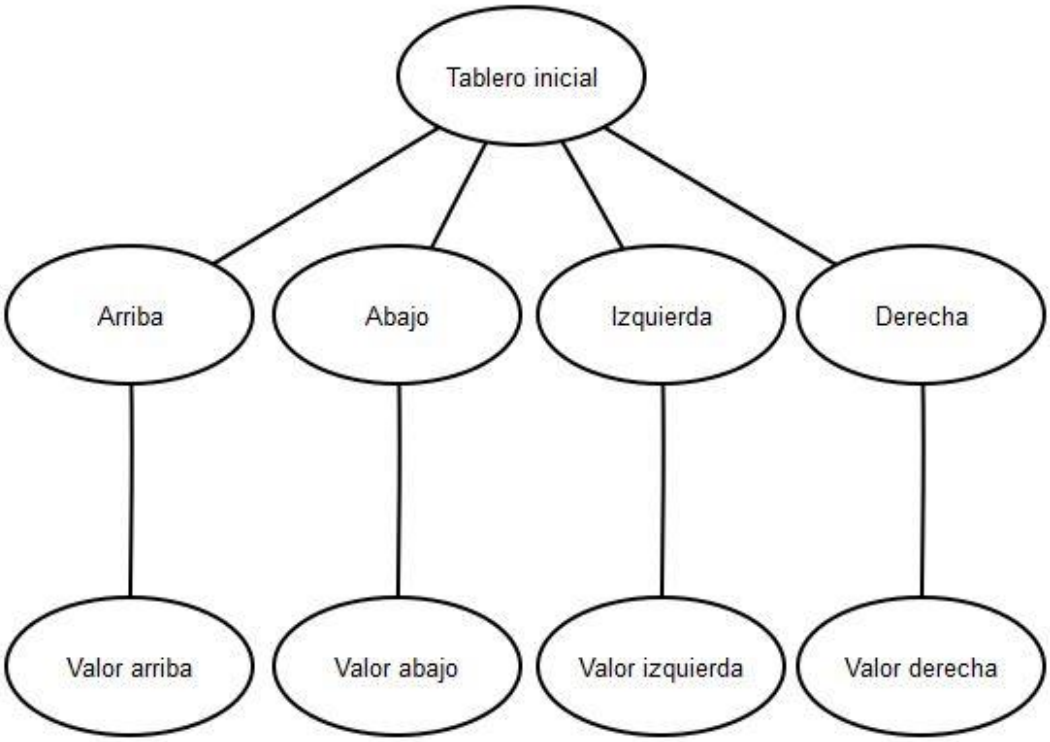


Ilustración 32 Simulación Monte Carlo

Este método se repetirá un número predeterminado de veces, y en cada paso se recogerá el valor obtenido y se añadirá a la exploración del movimiento correspondiente.

- A partir de ahora, se realizarán simulaciones selectivas. Esto quiere decir que, en lugar de simular todos los movimientos, sólo se simularán aquellos que sean prometedores debido a las estadísticas recogidas anteriormente. Por lo tanto, se repetirá hasta cumplir una condición de parada (un tiempo determinado) el método anterior, pero cada vez se seleccionará el movimiento que maximice la fórmula:

$$x = valor(x) + C\,\frac{\sum valor(pruebas)}{nº\_pruebas}$$

- Cuando el criterio de parada se cumpla, normalmente un determinado intervalo de tiempo, se seleccionará el hijo con mayor valor "x" como movimiento a ejecutar.

Capítulo 5 Pruebas y resultados

El objetivo número 10 descrito en la página 39, del que se hablará en este capítulo, consiste en realizar una serie de pruebas de los algoritmos diseñados en los apartados 4.5,4.6 y 4.7 y comprobar su validez a la hora de resolver el puzle del "2048".

A continuación, se procederá a explicar estas pruebas y los resultados obtenidos. Estas pruebas fueron realizadas en un ordenador con las siguientes características:

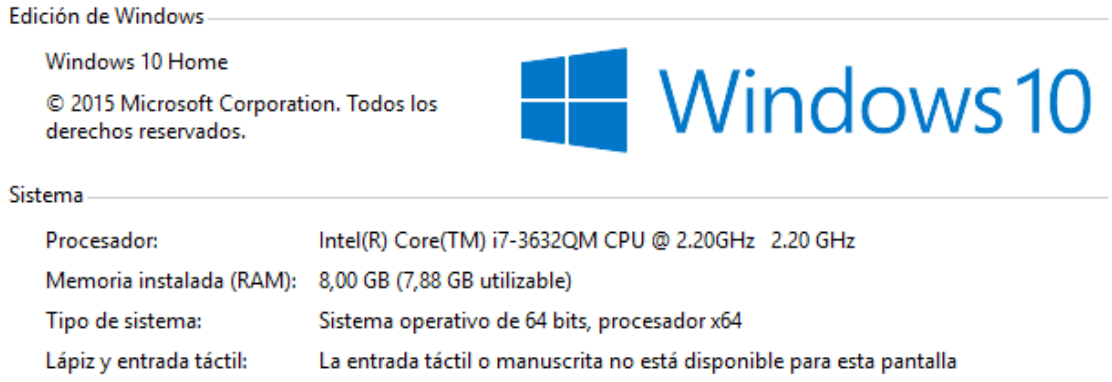


Ilustración 33 Especificaciones del PC

5.1. Pruebas previas

Una vez terminados los algoritmos, se comenzó a explotar de forma masiva cada uno de ellos para obtener resultados. Se hicieron 1000 pruebas con cada uno, así como unas pruebas que seleccionen los movimientos de forma aleatoria. Se obtuvieron los siguientes resultados:

Ficha mayor obtenida	Aleatorio	%
<=256	999	99,9
512	1	0,1
1024	0	0
2048	0	0
4096	0	0
Puntuación media	1.243,28	
Movimientos medios	137,863	
Tiempo medio	0,003676999	
Varianza tiempo	3,24E-05	

Tabla 57 Resultados de la prueba previa aleatoria

Ficha mayor obtenida	Evaluar	%
<=256	667	66,7
512	294	29,4
1024	39	3,9
2048	0	0
4096	0	0
Puntuación media	4.469,16	
Movimientos medios	359,495	
Tiempo medio	0,047796	
Varianza tiempo	0,000503524	

Tabla 58 Resultados de la prueba previa "Evaluar"

Ficha mayor obtenida	Evaluar Avanzado	%
<=256	88	8,8
512	236	23,6
1024	463	46,3
2048	207	20,7
4096	6	0,6
Puntuación media	15.251,732	
Movimientos medios	941	
Tiempo medio	0,134674	
Varianza tiempo	0,004521848	

Tabla 59 Resultados de la prueba previa "Evaluar Avanzado"

Ficha mayor obtenida	Árbol	%
<=256	86	8,6
512	174	17,4
1024	441	44,1
2048	292	29,2
4096	7	0,7
Puntuación media	17.246,724	
Movimientos medios	1.043,82	
Tiempo medio	0,678363	
Varianza tiempo	0,112571129	

Tabla 60 Resultados de la prueba previa "Árbol"

A continuación, se procederá a realizar comentarios sobre los resultados obtenidos:

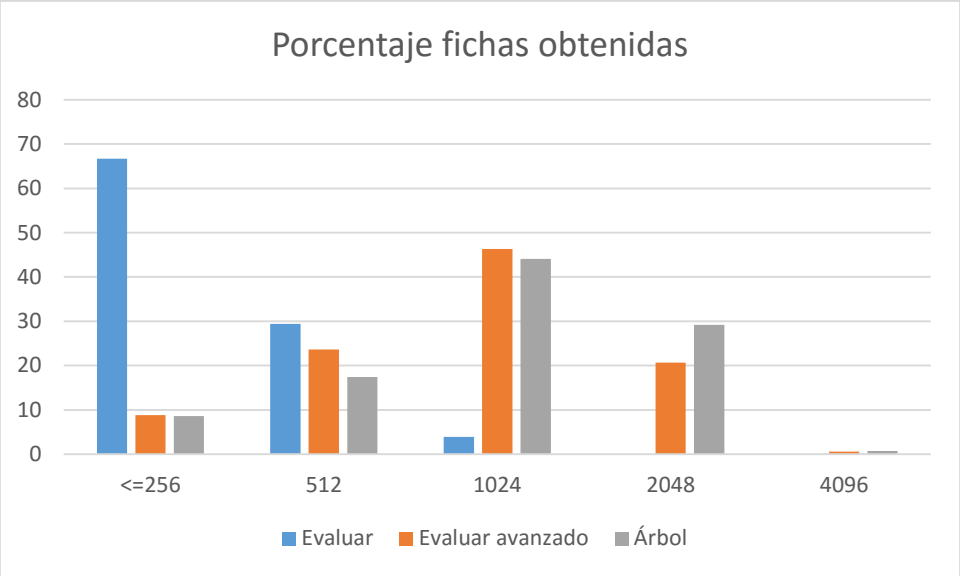


Ilustración 34 Porcentaje de piezas obtenidas en los algoritmos previos

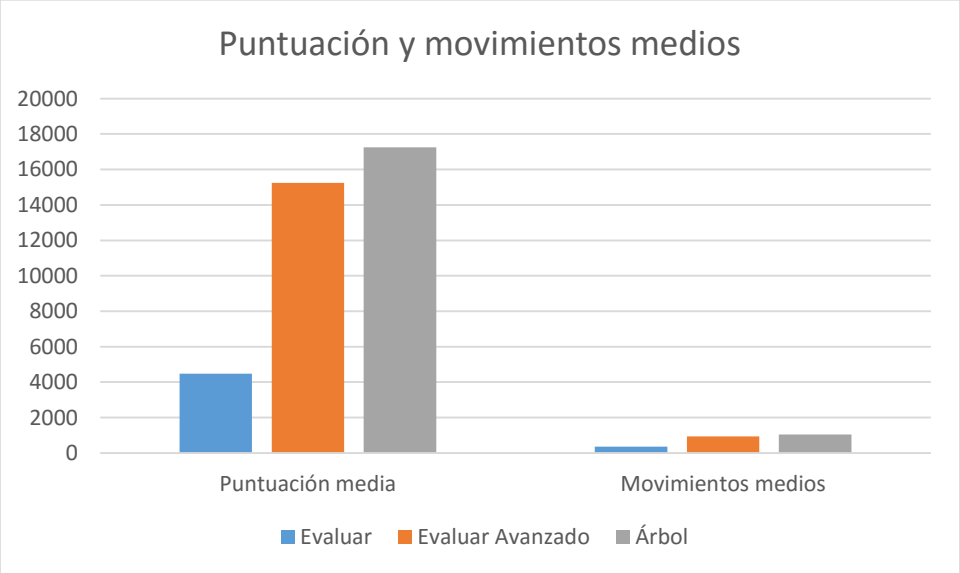


Ilustración 35 Puntuación y movimientos medios en los algoritmos previos

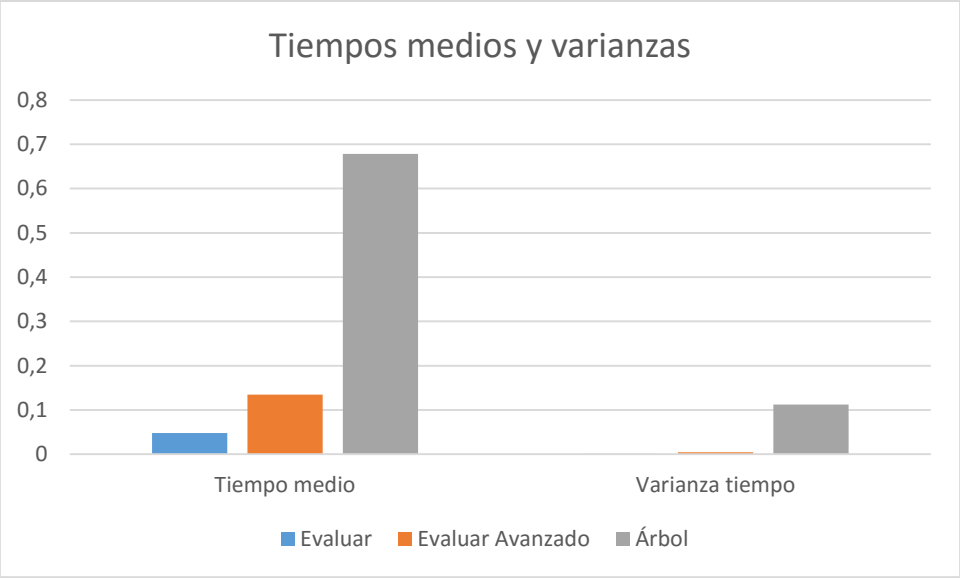


Ilustración 36 Tiempos medios y varianzas en los algoritmos previos

Sobre estos resultados obtenidos, se puede concluir que la técnica de árbol es muy superior al resto, siendo viable su explotación a mayor profundidad. Si bien es cierto que la resolución por medio de reglas puede ser muy potente, la complejidad en la definición de las mismas hace que se haya decantado por la otra alternativa. La obtención de mayor puntuación y la realización de mayor número de movimientos reafirman esta conjetura.

Aunque, como se puede apreciar, la diferencia de tiempo entre una y otra es notable, la posibilidad de obtener mejores resultados ampliando la profundidad utilizando los algoritmos diseñados de una manera muy sencilla, se decidió continuar con los experimentos con ella.

Con esto se resuelve la primera hipótesis de que por medio de técnicas de Inteligencia Artificial se puede resolver el juego del “2048” de mejor manera que con un jugador que elige piezas al azar.

5.2. Pruebas con árboles de búsqueda

5.2.1. Búsqueda realista

Concluidas las pruebas previas, se procedió a ejecutar los árboles de búsqueda, comenzando con el algoritmo "*Minimax*" para realizar búsquedas realistas. Debido a la gran cantidad de tiempo que requiere cada uno de ellos, se redujo el número de pruebas de 1000 a 100:

Profundidad 4			
Ficha mayor obtenida	Esquina %	Puntuación %	Serpiente%
<=256	90	8	1
512	10	53	20
1024	0	39	46
2048	0	0	32
4096	0	0	1
Puntuación media	2.831,8	9.314,64	19.384,24
Movimientos medios	256,83	635,83	1.154,06
Tiempo medio	1,62249	5,76408	11,39186
Varianza tiempo	0,14883293	2,36603466	21,8661454

Tabla 61 Resultados de búsqueda realista a profundidad 4

Profundidad 6			
Ficha mayor obtenida	Esquina %	Puntuación %	Serpiente%
<=256	93	1	0
512	7	14	9
1024	5	58	31
2048	0	27	49
4096	0	0	11
Puntuación media	3.112,16	17.745,52	27.592,28
Movimientos medios	278,3	1.080,14	1.534,53
Tiempo medio	31,17926	186,59502	275,31277
Varianza tiempo	43,4154683	2.540,40358	13.774,1432

Tabla 62 Resultados de búsqueda realista a profundidad 6

Profundidad 8			
Ficha mayor obtenida	Esquina %	Puntuación %	Serpiente%
<=256	54	0	1
512	42	0	3
1024	4	22	16
2048	0	72	53
4096	0	6	27
Puntuación media	5.217,04	30.259,08	36.647,88
Movimientos medios	406,87	1.688,63	1.958,5
Tiempo medio	769,8453	5.481,93914	6.326,96956
Varianza tiempo	23.146,7335	1.345.336,66	5.059.811,36

Tabla 63 Resultados de búsqueda realista a profundidad 8

No se han realizado pruebas a mayor profundidad debido a que una única prueba a una profundidad de 10 tardaba casi un día entero en realizarse.

Expresados estos resultados de forma gráfica:

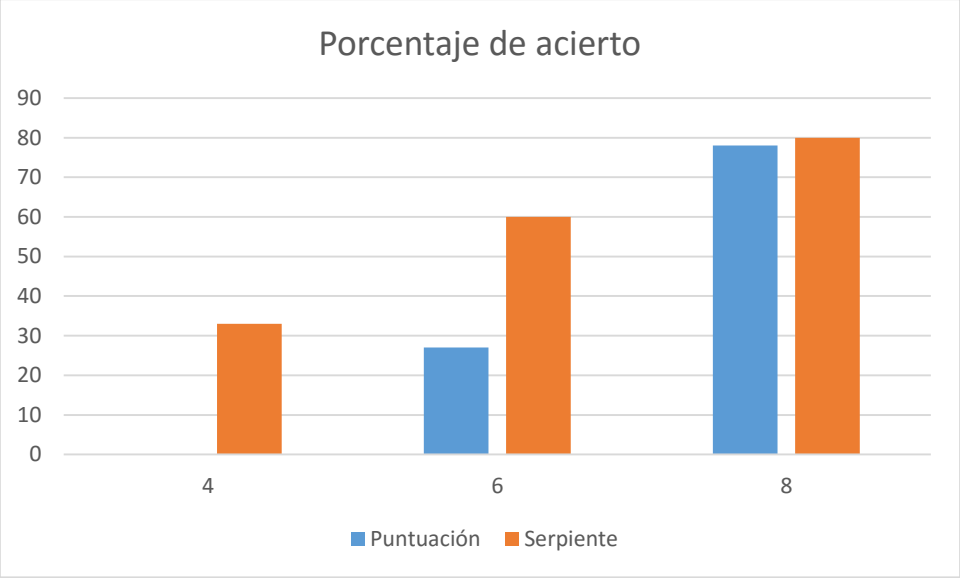


Ilustración 37 Porcentaje acierto en búsqueda realista

No se tuvo en cuenta la función de evaluación "Esquina", debido a que en todos los casos el porcentaje es del 0%.

Se puede apreciar el aumento sistemático del índice de acierto a medida que se incrementa la profundidad. A profundidad 8, el índice de acierto de ambas funciones de evaluación es muy similar, por lo que será necesario el estudio de otros factores para determinar cuál de ellas obtiene mejores resultados.

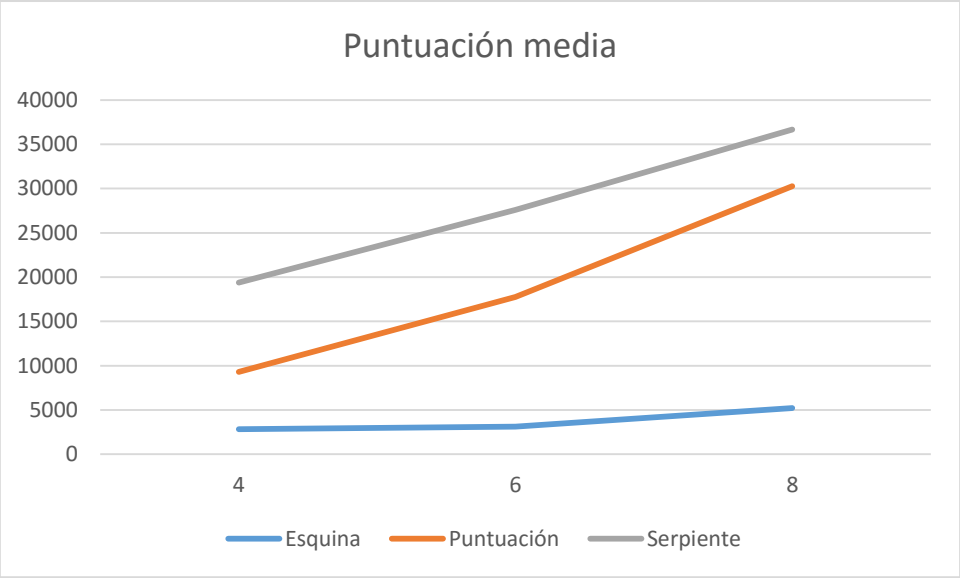


Ilustración 38 Puntuación media en búsqueda realista

Cabe destacar que la puntuación de la función de evaluación "Esquina" es muy inferior a la obtenida por las otras dos. Sobre el resto, las puntuaciones señalan claramente a la función de evaluación "Serpiente" como superior a la "Puntuación", estando siempre por encima en puntos medios obtenidos.

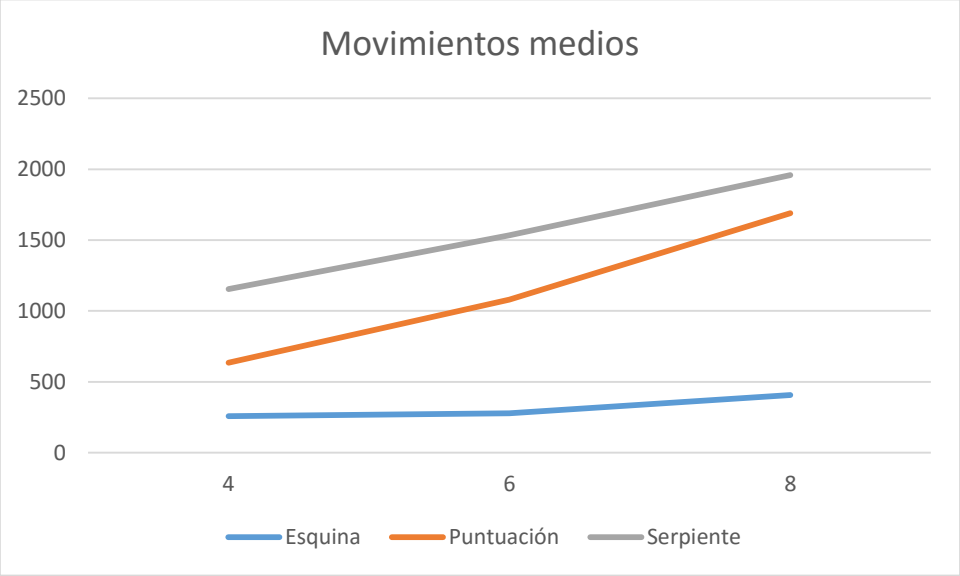


Ilustración 39 Movimientos medios en búsqueda realista

De manera complementaria a la anterior gráfica, la función de evaluación "Serpiente" ha realizado más movimientos medios que sus rivales.

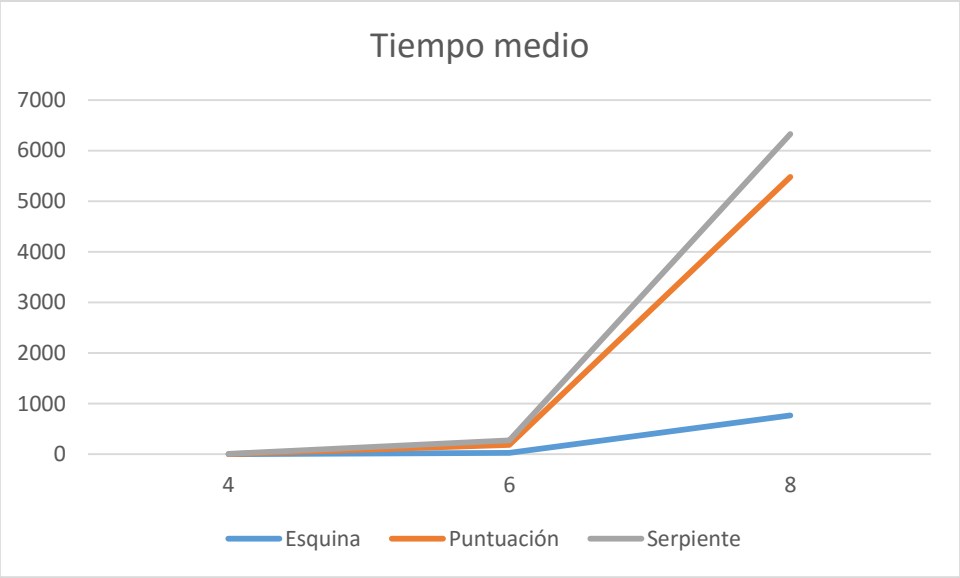


Ilustración 40 Tiempo medio en búsqueda realista

Es lógico pensar que cuantos más movimientos se realicen, más tiempo tardará el juego en terminar. A pesar de esta hipótesis, la diferencia de tiempos entre las dos funciones de evaluación no es muy significativa (a pesar de que la función de evaluación "Serpiente" tarda un poco más en terminar el juego).

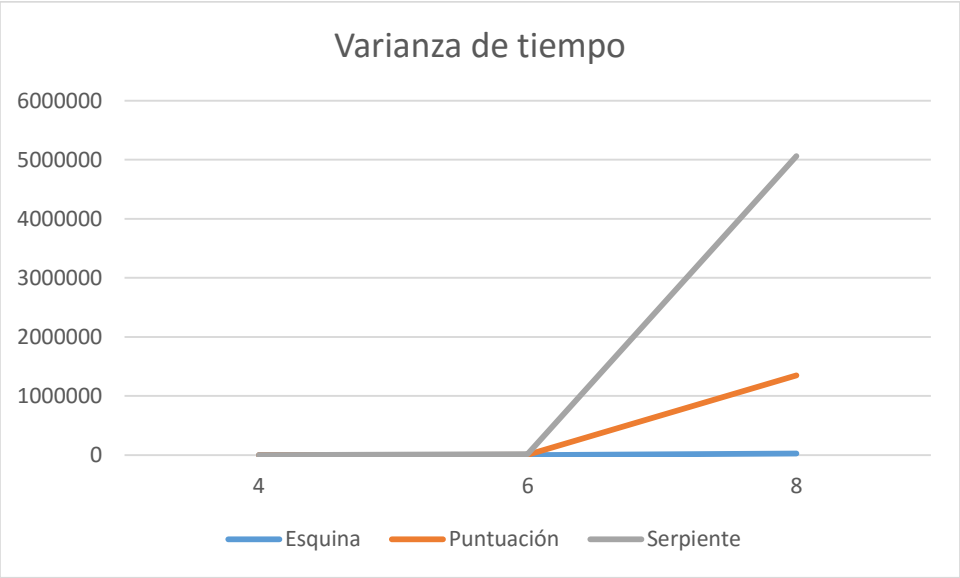


Ilustración 41 Varianza tiempo en búsqueda realista

En cuanto a las varianzas de tiempo, se puede apreciar que ha habido mucha más diferencia de tiempos en altas profundidades que en pruebas menos profundas.

La siguiente imagen muestra el resultado de la mejor partida de esta búsqueda, correspondiendo a la configuración correspondiente a profundidad 8 y función de evaluación “Serpiente”:

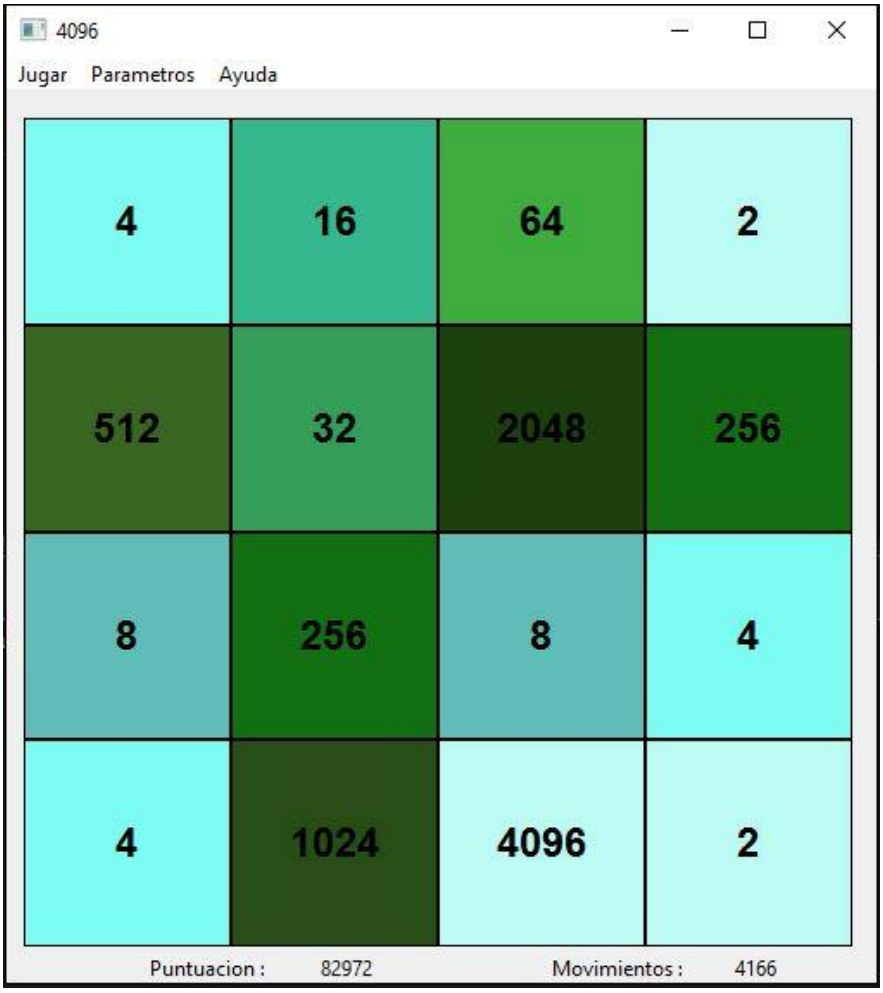


Ilustración 42 Mejor prueba con búsqueda realista

Como se ha podido apreciar en los resultados anteriores, a medida que la profundidad aumenta también mejoran los resultados, sobre todo en las funciones de evaluación “Serpiente” y “Puntuación”, siendo ésta última la mejor de las dos (36.647,88 de puntuación media frente a 30.259,08). El tiempo de ejecución también crece con la profundidad, y no sólo por el mayor número de movimientos realizados a mayores profundidades, sino también por la tardanza del algoritmo en realizar un único movimiento a altas profundidades.

La principal sorpresa fue el nulo porcentaje de acierto de la función de evaluación “Esquina”, como se ha podido comprobar en las Tablas Tabla 61, Tabla 62 y Tabla 63, así como en la Ilustración 37. En las siguientes búsquedas se espera que esta función de evaluación obtenga mejores resultados.

5.2.2. Búsqueda pesimista

A continuación, se mostrarán los resultados de las pruebas obtenidos por el algoritmo "α- β" a la hora de hacer búsquedas pesimistas:

Profundidad 4			
Ficha mayor obtenida	Esquina %	Puntuación %	Serpiente %
<=256	95	12	1
512	5	60	14
1024	0	28	29
2048	0	0	43
4096	0	0	13
8192	0	0	0
Puntuación media	2.672,88	8.205,8	27.847,16
Movimientos medios	249,52	573,26	1.563,47
Tiempo medio	1,25401	3,17109	11,91442
Varianza tiempo	0,11613086	0,73436042	33,160554

Tabla 64 Resultados de búsqueda pesimista a profundidad 4

Profundidad 6			
Ficha mayor obtenida	Esquina %	Puntuación %	Serpiente %
<=256	91	2	2
512	9	29	3
1024	0	64	18
2048	0	5	42
4096	0	0	34
8192	0	0	1
Puntuación media	3.287,88	13.145,04	42.188
Movimientos medios	289,75	846,06	2.239,23
Tiempo medio	13,82228	39,75809	189,3512
Varianza tiempo	12,8870315	104,905492	6.915,741

Tabla 65 Resultados de búsqueda pesimista a profundidad 6

Profundidad 8			
Ficha mayor obtenida	Esquina %	Puntuación %	Serpiente %
<=256	83	1	0
512	17	15	0
1024	0	64	11
2048	0	20	44
4096	0	0	39
8192	0	0	6
Puntuación media	4.103,2	17.206,12	51.142,8
Movimientos medios	343,35	1.056,17	2.632,53
Tiempo medio	120,77807	378,36625	2.112,37091
Varianza tiempo	906,298874	10.934,0338	572.011,727

Tabla 66 Resultados de búsqueda pesimista a profundidad 8

Como en las pruebas anteriores, la máxima profundidad viable es 8, puesto que, al aumentar más allá de la misma, el tiempo de ejecución de una sola prueba era enorme.

Expresados los resultados de forma gráfica:

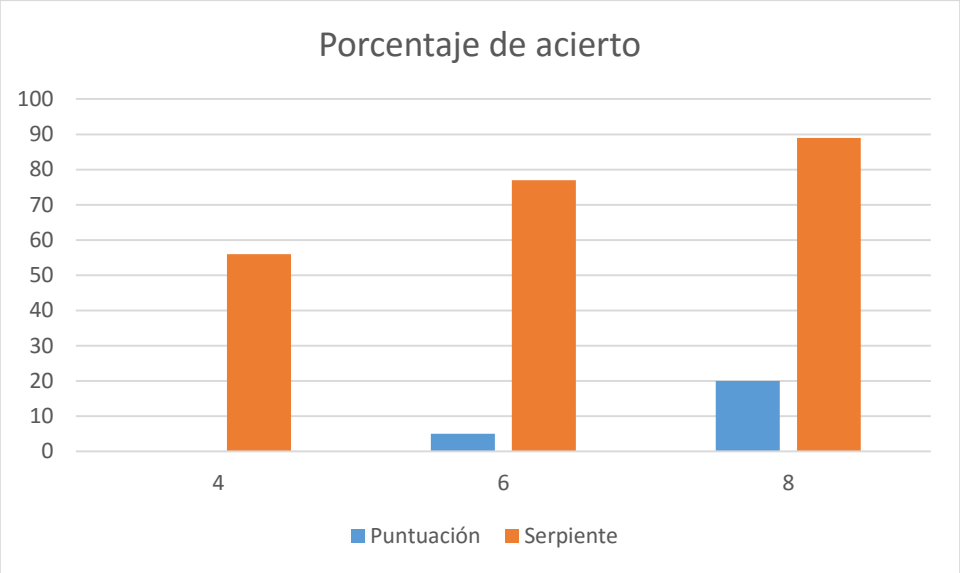


Ilustración 43 Porcentaje de acierto en búsqueda pesimista

Una vez más, se considera acierto el obtener una ficha 2048 o superior. No se tuvo en cuenta la función de evaluación "Esquina", debido a que en todos los casos el porcentaje es del 0%.

Como sucedió en las pruebas anteriores, la función de evaluación "Esquina" no ha conseguido alcanzar la meta de obtener una ficha 2048 en ninguna prueba. Como diferencia a la búsqueda realista, la función de evaluación "Serpiente" ha alcanzado un porcentaje de acierto muy alto comparado con sus rivales, incluyendo la función de evaluación "Puntuación" que obtuvo resultados similares en la anterior búsqueda. Esta diferencia de porcentaje de éxito se prevé que será determinante en el resto de comparaciones.

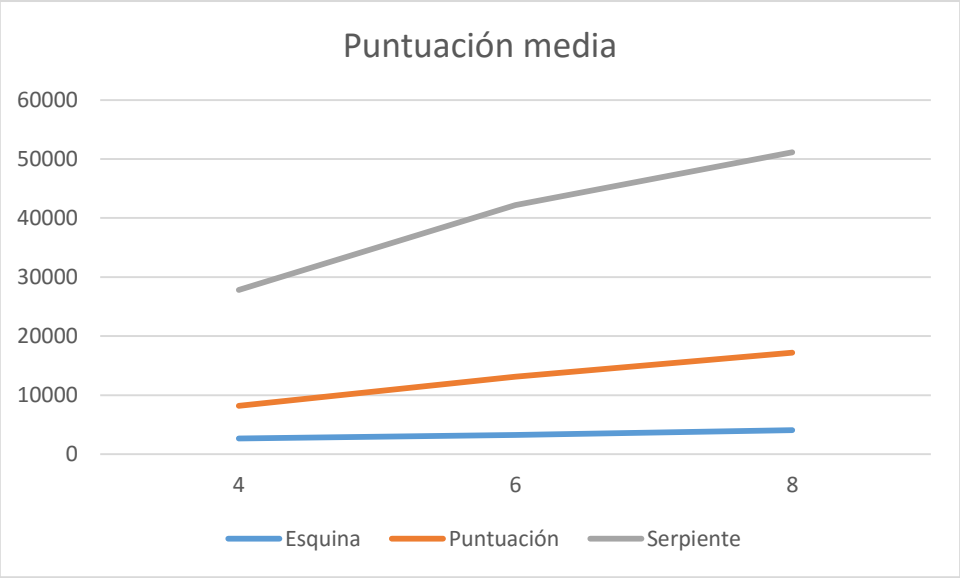


Ilustración 44 Puntuación media en búsqueda pesimista

Como se dedujo en la gráfica anterior, las puntuaciones obtenidas por la función de evaluación "Serpiente" son muy altas, casi triplicando a la segunda mejor y superando incluso la mejor puntuación obtenida en la anterior búsqueda (51.142,8 en búsqueda pesimista y 36.647,88 en búsqueda realista).

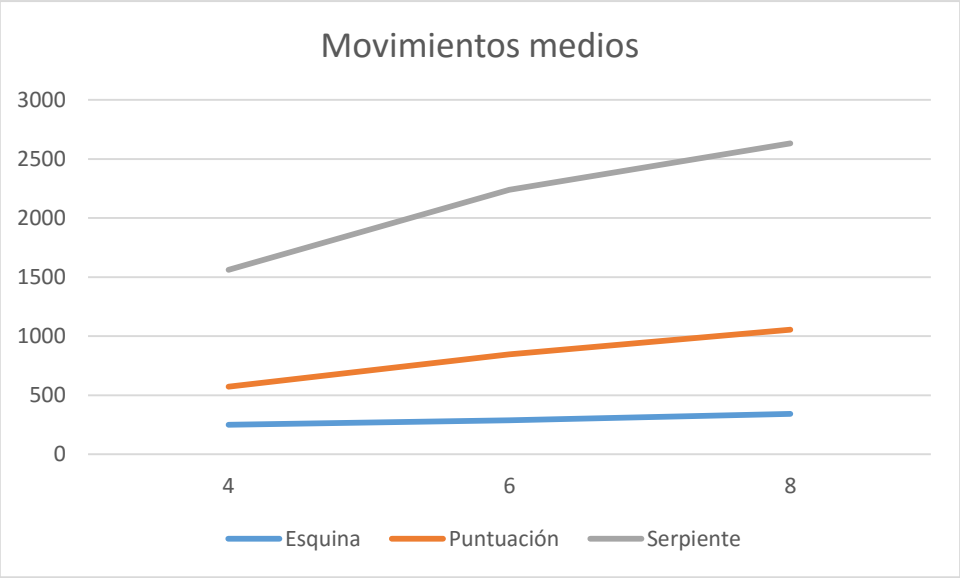


Ilustración 45 Movimientos medios en búsqueda pesimista

Una vez más, se demuestra que las altas puntuaciones implican un gran número de movimientos. De forma idéntica al análisis de las puntuaciones anteriores, los movimientos realizados por la función de evaluación "Serpiente" superan con creces a los de las otras funciones de evaluación.

Ilustración 46 Tiempo medio en búsqueda pesimista

El tiempo medio que ha necesitado la función de evaluación "Serpiente" es, como cabe esperar, muy superior al necesitado por las otras funciones de evaluación.

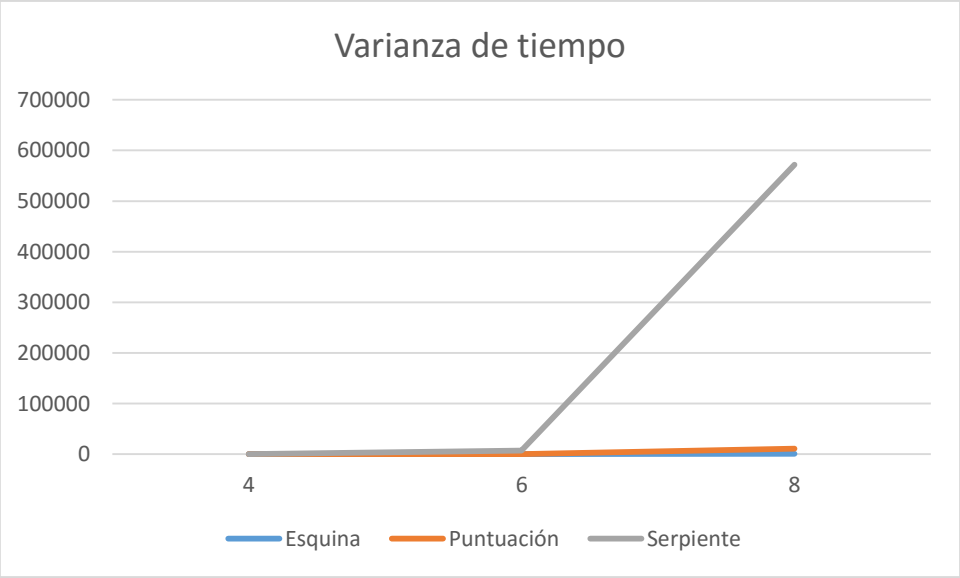


Ilustración 47 Varianza de tiempo en búsqueda pesimista

Como se puede deducir por los otros análisis realizados en este tipo de búsqueda, la varianza encontrada en la función de evaluación "Serpiente" es muy alta. En el resto, la diferencia es mínima en comparación con la vencedora.

La siguiente ilustración mostrará el mejor resultado de este tipo de búsqueda. Los parámetros que obtuvieron este tablero son la función de evaluación “Serpiente” y profundidad 8.

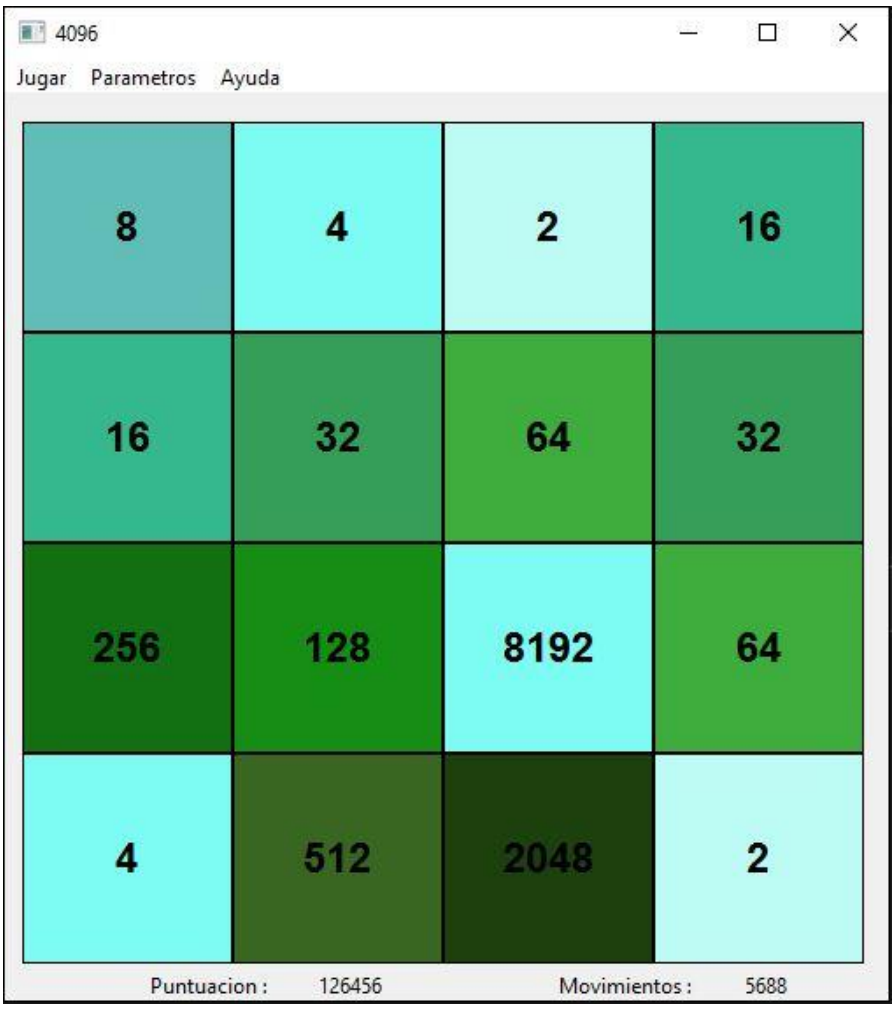


Ilustración 48 Mejor prueba con búsqueda pesimista

Como se puede apreciar, en esta partida se obtuvo una ficha de valor 8192, para la cual es necesario conseguir y juntar cuatro fichas 2048. Además, ha podido juntar piezas hasta alcanzar una ficha 2048 adicional, por lo que se puede decir que en este caso ha alcanzado la meta planteada cinco veces.

Con respecto a los resultados obtenidos en este tipo de búsqueda, se puede ver que existe, una vez más, una mayor diferencia en las puntuaciones y movimientos a medida que se va incrementando la profundidad, siendo la función de evaluación “Serpiente” la que consigue mejores partidas de forma claramente diferenciada. Esto hace que esta función de evaluación sea la preferida hasta ahora, habiendo sido la ganadora de este tipo de búsqueda como del anterior.

Una vez más, la “Esquina” sigue sin ser capaz de alcanzar el objetivo del juego. Es viable pensar que la estrategia seguida por la misma no es compatible con estos tipos de búsqueda, por lo que se tratará de obtener mejores resultados en las siguientes.

5.2.3. Búsqueda optimista

Como ya se ha mencionado en el capítulo 4, se presentó la hipótesis de que realizar búsquedas optimistas supondría la asunción de excesivos riesgos que perjudicarían a la hora de obtener buenos resultados.

Si bien no se han realizado pruebas específicas para demostrar esta conjetura, los resultados parecen reforzarla. La importante diferencia entre no asumir riesgos y asumir unos pocos a favor del primer tipo de búsqueda lleva a pensar que el realizar movimientos de manera muy optimista no servirá para realizar grandes partidas.

5.3. Pruebas con el algoritmo de Monte Carlo

Cabe destacar que se hicieron pequeñas pruebas para determinar qué parámetros eran los óptimos para la resolución de este algoritmo, concretamente el valor de *C* y el criterio de selección de movimientos. Los resultados de estas pruebas son:

C			
	C=0,75	**C=1**	**C=1,25**
Puntuación media	35581,12	37097,6	40341,44
Movimientos medios	1959,24	2025,64	2174,16

Tabla 67 Resultados de la prueba sobre la variable C del algoritmo de Monte Carlo

Criterio de selección			
	Aleatorio	**Probabilística**	**Directo**
Puntuación media	27165,12	35604,16	40341,44
Movimientos medios	1546,72	1949,84	2174,16

Tabla 68 Resultados de la prueba sobre el criterio de selección algoritmo de Monte Carlo

Expresados de forma gráfica:

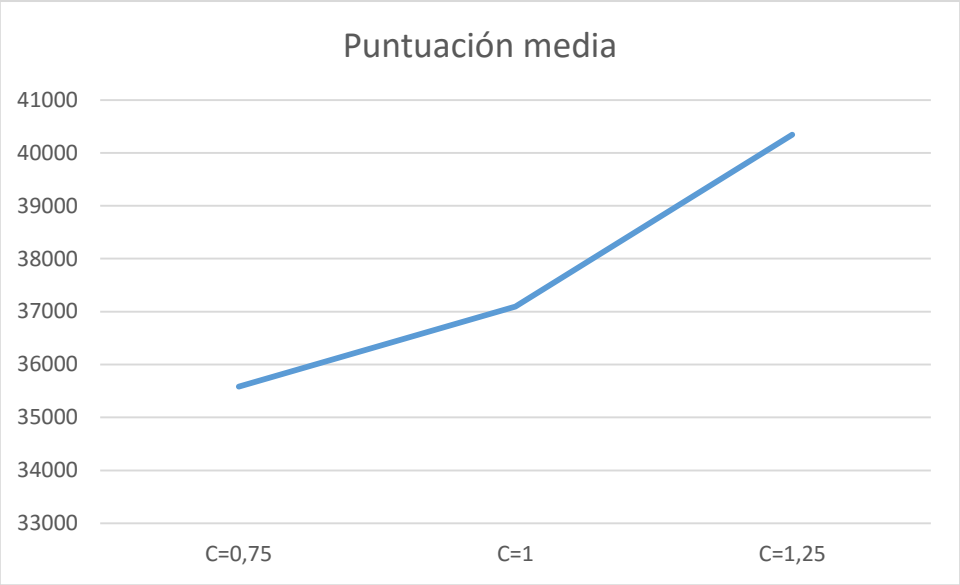


Ilustración 49 Puntuación media de las pruebas sobre la variable C

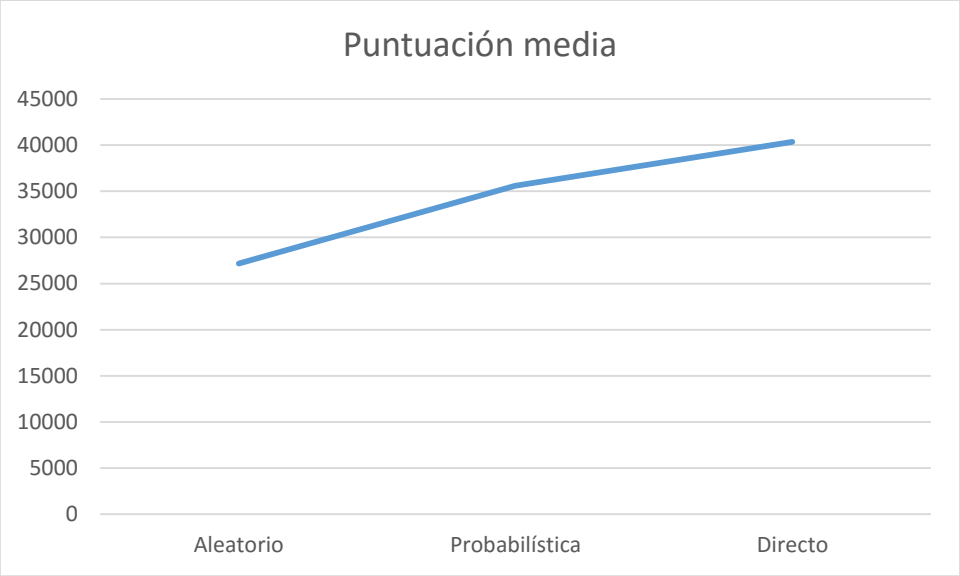


Ilustración 50 Puntuación media de las pruebas sobre el criterio de selección

Con lo cual, se tomará la siguiente configuración para la ejecución de las pruebas definitivas.

- *C* igual a 1,25.
- Criterio de selección directo.

A continuación, se mostrarán los resultados obtenidos con esta configuración.

Profundidad 10			
Ficha mayor obtenida	Esquina %	Puntuación %	Serpiente %
<=256	32	0	32
512	63	4	63
1024	5	23	5
2048	0	66	0
4096	0	7	0
Puntuación media	5.538,88	30.091,32	5.274,76
Movimientos medios	417,09	1.682,98	396,3
Tiempo medio	512,79056	1.983,67816	498,85459
Varianza tiempo	22.436,04458	405.626,0332	23.032,06538

Tabla 69 Resultados de búsqueda con Monte Carlo a profundidad 10

Profundidad 20			
Ficha mayor obtenida	Esquina %	Puntuación %	Serpiente %
<=256	40	0	34
512	56	0	59
1024	4	7	7
2048	0	83	0
4096	0	10	0
Puntuación media	5.077,12	35.622,56	5.251,56
Movimientos medios	387,95	1.954,37	393,54
Tiempo medio	554,80616	2.614,18878	583,74247
Varianza tiempo	35.555,5426	330.818,013	42.503,08931

Tabla 70 Resultados de búsqueda con Monte Carlo a profundidad 20

Profundidad 40			
Ficha mayor obtenida	Esquina %	Puntuación %	Serpiente %
<=256	24	0	41
512	69	1	56
1024	7	29	3
2048	0	69	0
4096	0	1	0
Puntuación media	5.629,6	27.305,04	4.706,72
Movimientos medios	416,12	1548,56	360,67
Tiempo medio	750,55578	2.503,49588	666,54931
Varianza tiempo	52.925,30691	366.125,1448	52.871,2903

Tabla 71 Resultados de búsqueda con Monte Carlo a profundidad 40

Profundidad 60			
Ficha mayor obtenida	Esquina %	Puntuación %	Serpiente %
<=256	21	0	35
512	70	4	57
1024	9	42	8
2048	0	47	0
4096	0	10	0
Puntuación media	5.927,08	24.097,56	5.169,44
Movimientos medios	433,94	1394,61	384,85
Tiempo medio	892,6719	2.455,72039	810,07291
Varianza tiempo	65.144,79279	457.690,9239	84.047,6202

Tabla 72 Resultados de búsqueda con Monte Carlo a profundidad 60

Profundidad 80			
Ficha mayor obtenida	Esquina %	Puntuación %	Serpiente %
<=256	21	0	35
512	70	4	57
1024	9	42	8
2048	0	47	0
4096	0	10	0
Puntuación media	5.927,08	24.097,56	5.169,44
Movimientos medios	433,94	1394,61	384,85
Tiempo medio	892,6719	2.455,72039	810,07291
Varianza tiempo	65.144,79279	457.690,9239	84.047,6202

Tabla 73 Resultados de búsqueda con Monte Carlo a profundidad 80

Expresados de forma gráfica:

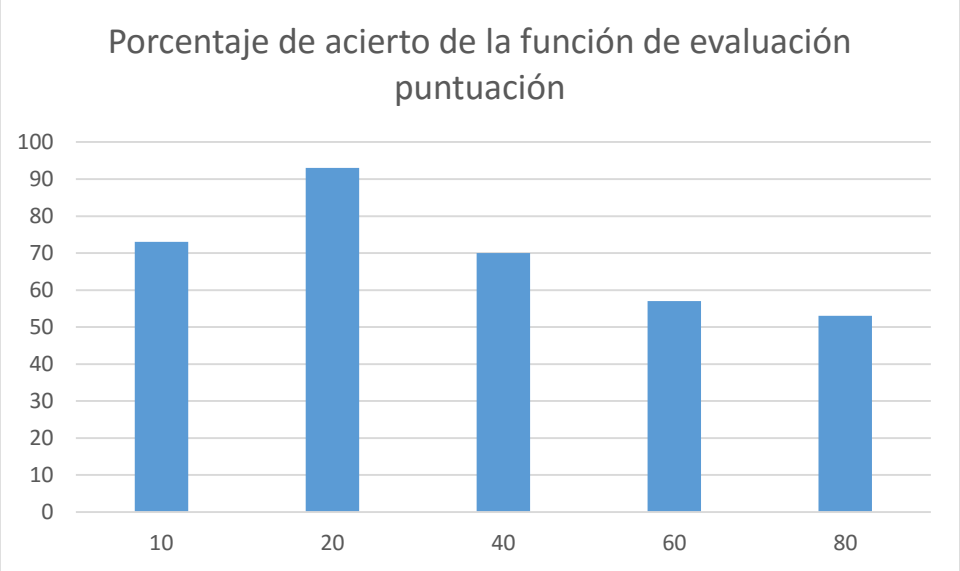


Ilustración 51 Porcentaje de acierto de la función de evaluación puntuación en búsqueda con Monte Carlo

Exceptuando la función de evaluación “Puntuación”, ninguna de ellas ha conseguido alcanzar la meta pretendida. En cambio, la función de evaluación menos informada ha obtenido buenas puntuaciones en este tipo de búsqueda, consiguiendo el mejor porcentaje de acierto hasta ahora (93% frente al 89% de la búsqueda pesimista).

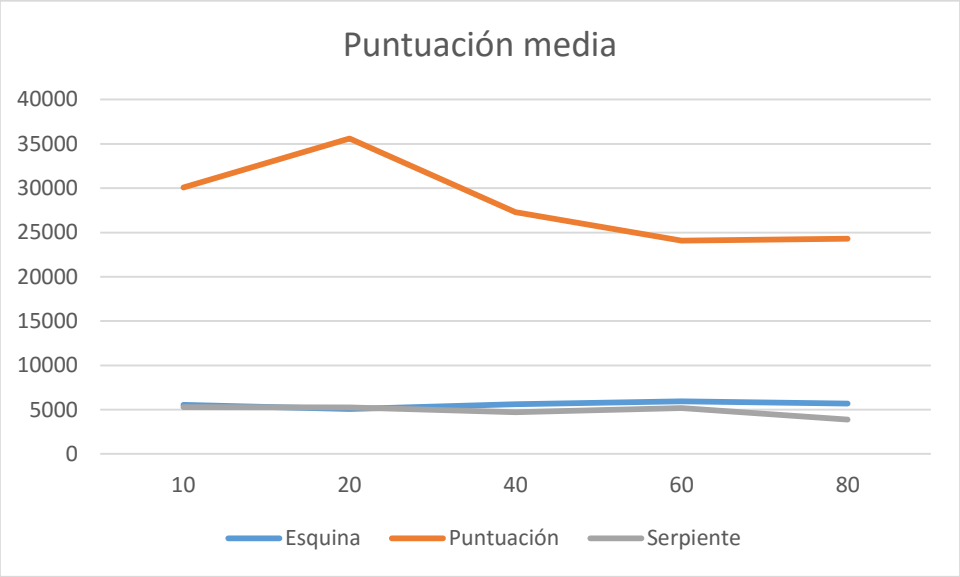


Ilustración 52 Puntuación media en búsqueda con Monte Carlo

De manera contraria a las búsquedas con algoritmos deterministas, al parecer se consiguen peores puntuaciones a medida que se incrementa la profundidad. Se ha obtenido un incremento de la puntuación con profundidad 20, lo que se puede deducir que puede ser una profundidad lo suficientemente alta como para obtener buenas puntuaciones, pero no tanto como para tener demasiada anticipación. Se recuerda que este método no tiene en cuenta el azar de forma tan precisa como los métodos deterministas, por lo que un exceso de profundidad parece que perjudica al tomar un camino sin tener en cuenta todas las posibilidades.

Puede ser por esta razón que la función de evaluación “Puntuación” sea tan superior a las otras dos en este tipo de búsqueda. Las otras, al tener la estrategia integrada en el cálculo de su evaluación, son muy dependientes del buen manejo del azar. Al no existir este manejo en este tipo de búsqueda, ambas obtienen resultados peores.

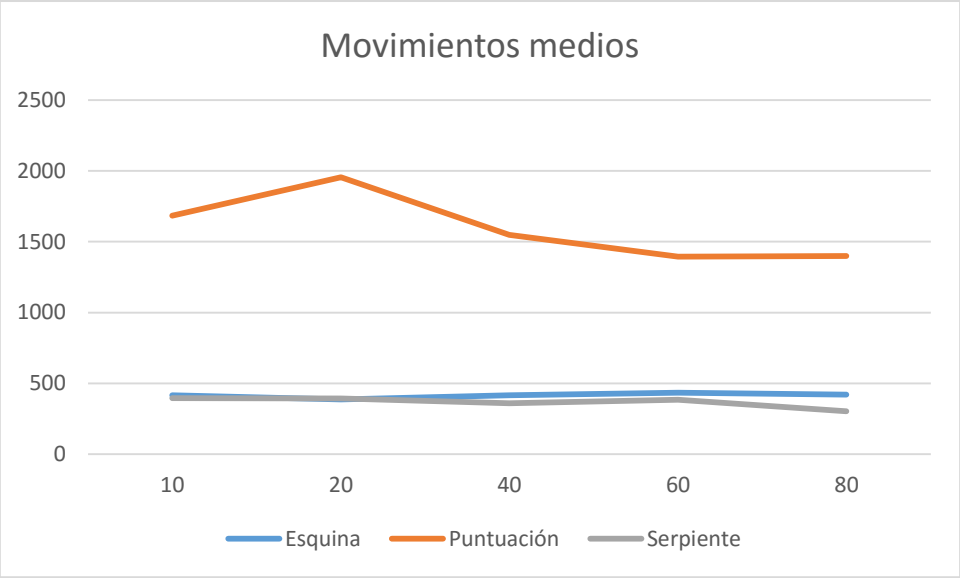


Ilustración 53 Movimientos medios en búsqueda con Monte Carlo

Como era de esperar, los movimientos obtenidos con la función de evaluación "Puntuación" son mucho más altos que los que han obtenido las otras dos.

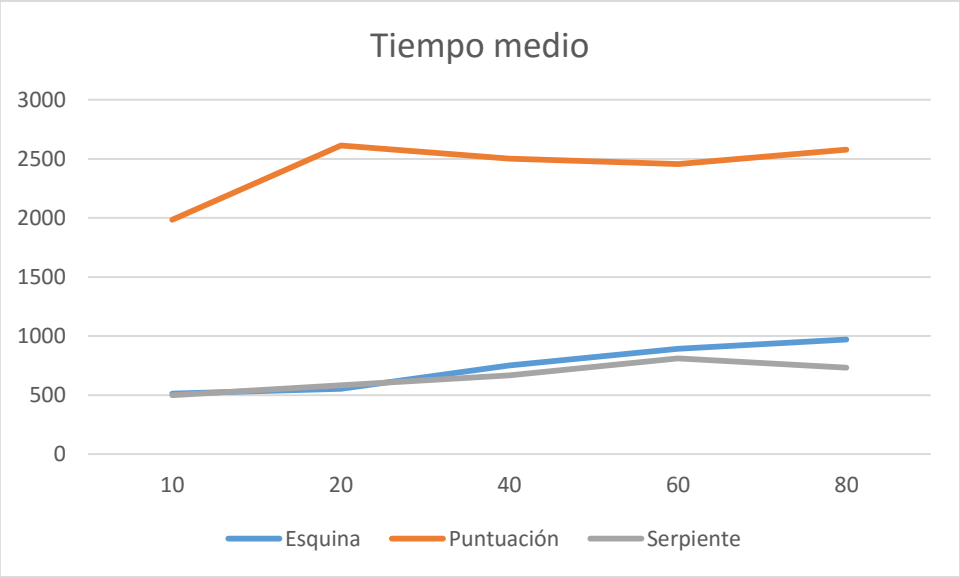


Ilustración 54 Tiempo medio en búsqueda con Monte Carlo

Lo mismo ocurre con los tiempos necesarios para realizar cada partida.

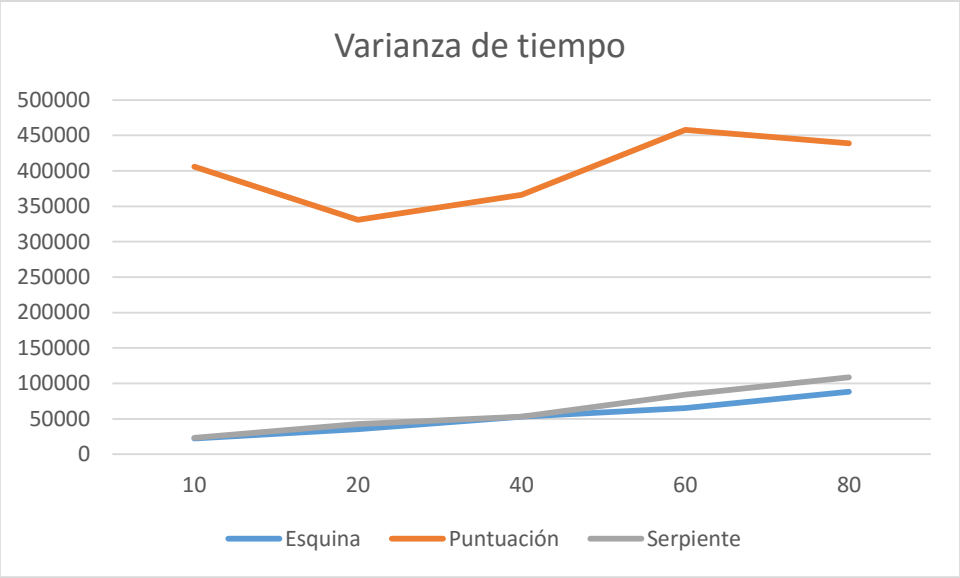


Ilustración 55 Varianza de tiempo en búsqueda con Monte Carlo

Y con las varianzas.

Este es el mejor resultado obtenido en esta búsqueda. Utilizó la función de evaluación "Puntuación" a una profundidad de 20:

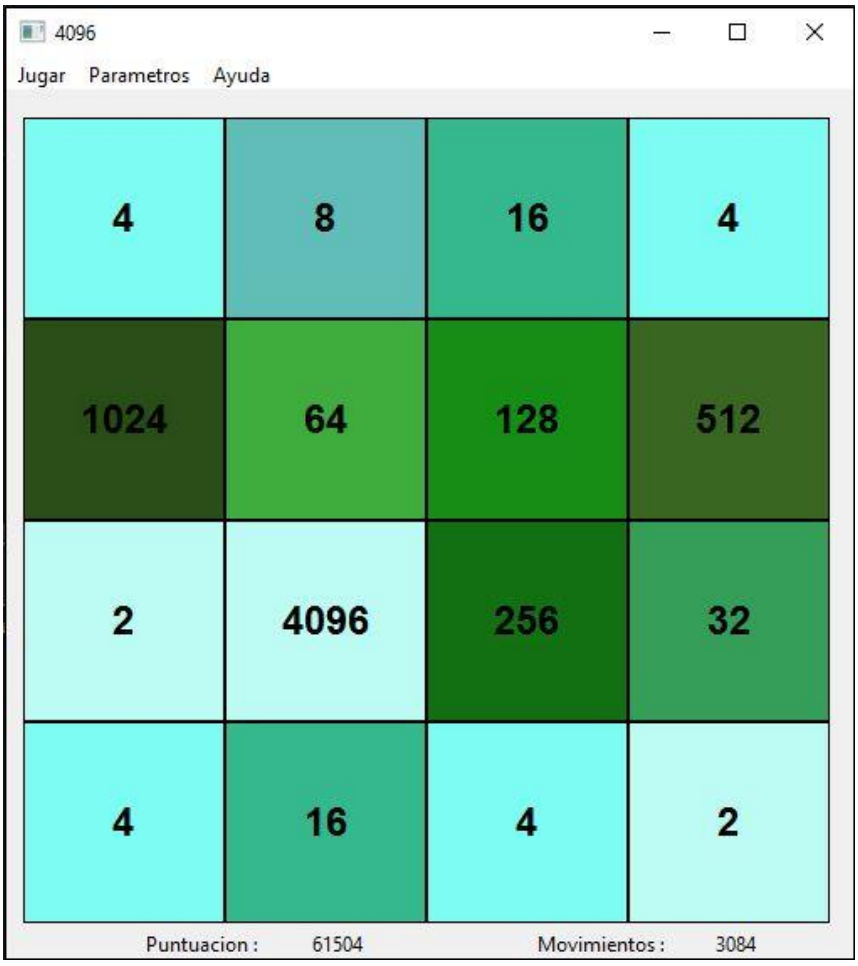


Ilustración 56 Mejor prueba con búsqueda con Monte Carlo

Es destacable señalar el fracaso tanto de la función de evaluación “Serpiente” (que tan buenos resultados había obtenido en las anteriores pruebas) como de la función de evaluación “Esquina” a la hora de realizar búsquedas no deterministas.

Sin embargo, la otra función de evaluación, correspondiente a la puntuación, obtuvo un porcentaje de acierto sorprendentemente alto, superando todos los obtenidos hasta ahora. Si bien su puntuación no consigue superar al resto de forma tan drástica, si el objetivo principal fuera obtener una pieza de 2048 ésta es la mejor opción.

5.4. Conclusiones sobre los resultados

A continuación, se presentará una gráfica sobre la evolución de estas pruebas. En ellas, se reflejará el mejor resultado en cada caso, tanto en cuanto a puntuación como en cuanto a porcentaje de acierto:

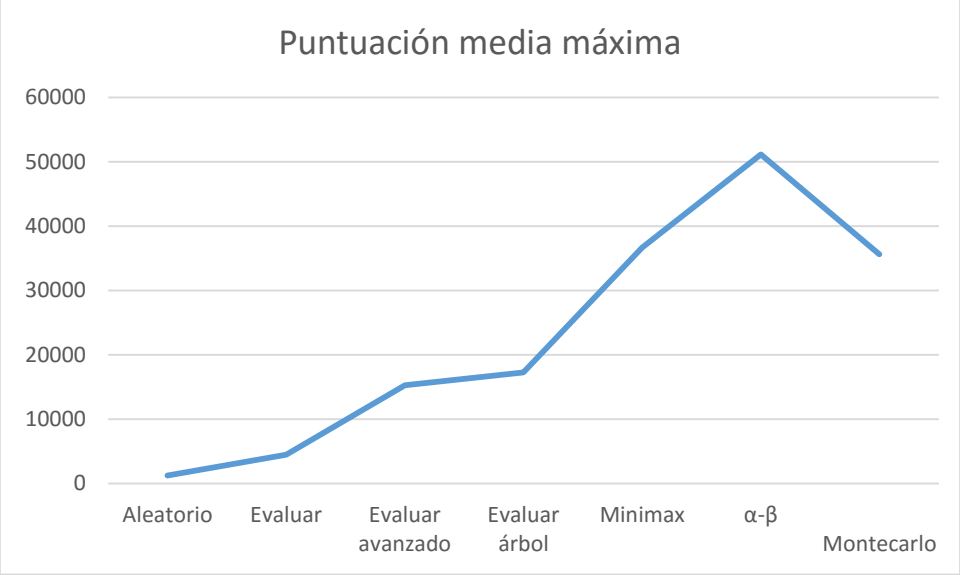


Ilustración 57 Puntuación media máxima

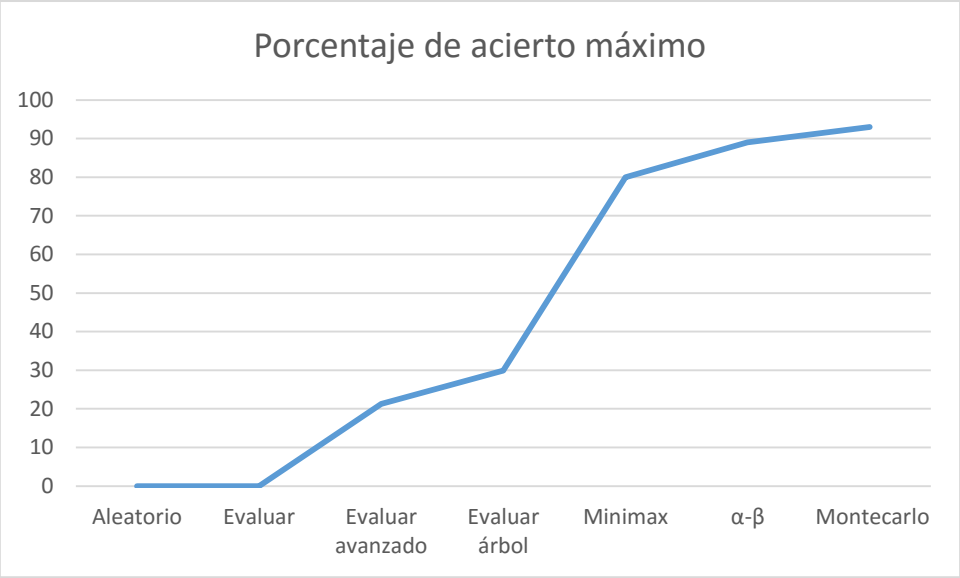


Ilustración 58 Porcentaje de acierto máximo

Como se puede apreciar, las pruebas revelan una alta puntuación y porcentaje de acierto, sobre todo en las dos últimas técnicas. Sin embargo, se puede concluir que el algoritmo α-β es el mejor de los utilizados en este proyecto, debido a su gran puntuación, gran porcentaje de acierto y la sencillez de mejora del funcionamiento del propio algoritmo con el aumento de la profundidad.

A continuación, se estudiará cuál de las funciones de evaluación ha sido la mejor para la resolución de este problema. Se han obtenido los siguientes resultados:

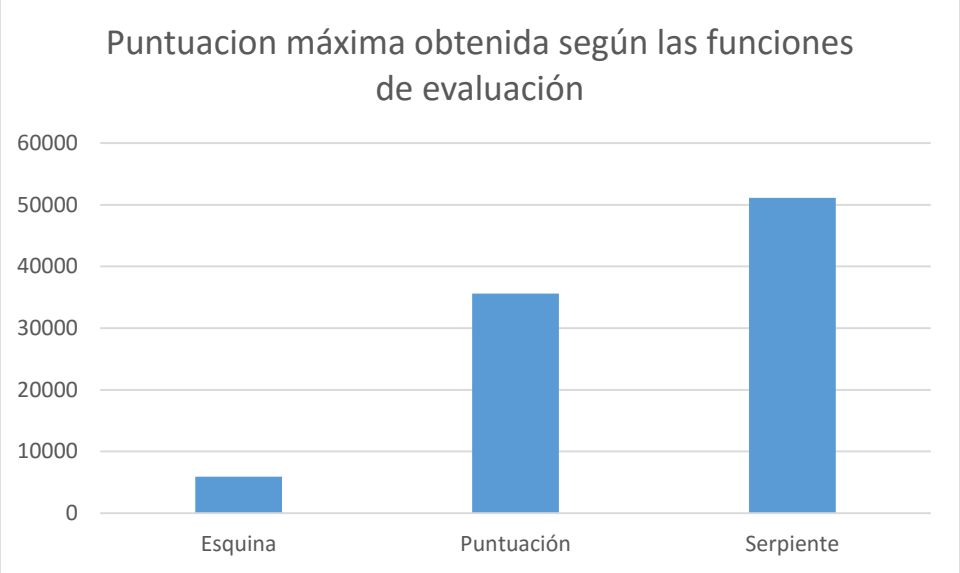


Ilustración 59 Puntuación máxima obtenida según las funciones de evaluación

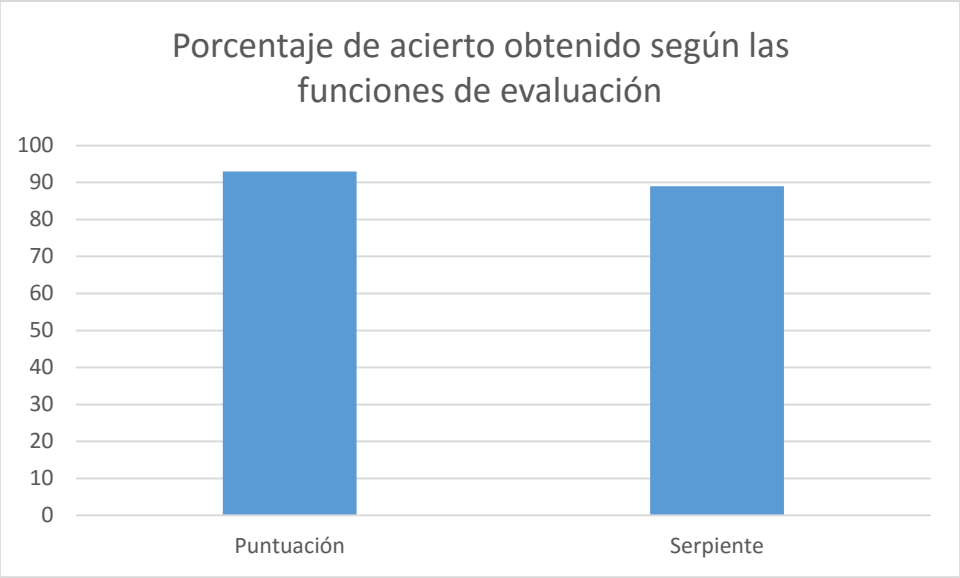


Ilustración 60 Porcentaje de acierto obtenido según las funciones de evaluación

Sobre las funciones de evaluación, se concluye que la primera de ellas, "Esquina", no sirve para la resolución de este problema en particular. Si bien la idea original puede ser acertada, es necesaria una reformulación para darle un uso práctico.

En cuanto a la segunda, "Puntuación", ha demostrado ser la mejor a la hora de emplear el algoritmo de Monte Carlo. La falta de información parece ser necesaria a la hora de ejecutar este algoritmo, puesto que el resto de funciones de evaluación fracasaron de forma notable.

Por último, la función de evaluación "Serpiente" obtuvo muy buenos resultados y puntuaciones en la resolución del problema con el algoritmo α-β. Si bien no ha conseguido un porcentaje de acierto superior que la función de evaluación "Puntuación", la importante diferencia en cuanto a puntuación es suficiente para determinar que esta función de evaluación es la mejor de las probadas.

Como conclusiones finales a estos resultados, remarcar que la mejor configuración de las ejecutadas para la resolución del problema del 2048 ha sido el algoritmo α-β utilizando la función de evaluación "Serpiente".

Las mejores tres configuraciones conseguidas han sido:

Tipo de búsqueda	Función	Profundidad	Porcentaje de acierto	Puntuación media	Tiempo medio (segundos)
Búsqueda pesimista	Serpiente	8	89	51.142,8	2.112,37091
Búsqueda con Monte Carlo	Puntuación	20	93	35.622,56	2.614,18878
Búsqueda realista	Serpiente	8	80	36.647,88	6.326,96956

Tabla 74 Top mejores configuraciones

Estos resultados demuestran que los métodos deterministas, al tener más control del factor aleatorio, obtiene mejores resultados que los métodos no deterministas, que se centran en tomar decisiones a mayor plazo teniendo en cuenta muchos más movimientos.

A modo de conclusión final, recordar la meta de alcanzar más de 20.000 puntos que obtenían los jugadores humanos. Sin embargo, y como era de esperar, ninguna de las pruebas ha conseguido superar el controlador explicado en la sección 2.5.3, página 35.

Capítulo 6 Conclusiones y líneas futuras

A modo de conclusión final de este proyecto, se repasarán los objetivos planteados en el capítulo 3 de esta memoria con el propósito de comprobar la resolución de los mismos.

6.1. Objetivos alcanzados

A continuación, se listarán los objetivos pretendidos de este proyecto y se comentará si se han conseguido alcanzar:

- ✓ **Análisis de un programa para la resolución del juego "2048":** Como se ha visto en el capítulo 4, sección 4.1, página 43 se ha diseñado y analizado de forma correcta el programa que se ha realizado a lo largo del proyecto por medio de la definición de requisitos y casos de uso.

- ✓ **Diseño de un programa para la resolución del problema "2048":** Como se ha visto en el capítulo 4, sección 4.1, página 69 se ha diseñado de forma correcta el programa que se ha realizado a lo largo del proyecto por medio de un diagrama de clases.

- ✓ **Implementación de un programa para la resolución del puzle "2048":** Utilizando el análisis y el diseño especificado por los anteriores objetivos, se creó un programa que aunará todos esos requisitos, casos de uso y diagramas de clase.

- ✓ **Implementación de las reglas del juego:** Como paso previo a la ejecución a las pruebas, se implementaron las funciones que corresponden a las reglas del juego, como se especifica en las funciones de movimientos definidas en la página 70.

- ✓ **Implementación de una interfaz gráfica para ejecutar el puzle con distintas configuraciones:** Se ha diseñado y realizado una interfaz gráfica con los elementos propuestos, tal y como se especifica en el capítulo 4, sección 4.3, página 73.

- ✓ **Definición de funciones de evaluación:** En el capítulo 4, sección 4.4, página 75, se han definido una serie de funciones de evaluación que cada una integra distintas estrategias enfocadas en la resolución del puzle "2048".

- ✓ **Implementación de algoritmos básicos para la resolución del puzle:** Como se ha podido ver en el capítulo 4, sección 4.5, página 77, se han diseñado dos algoritmos basados en técnicas deterministas para la resolver el juego.

- ✓ **Implementación de algoritmos deterministas para la resolución del puzle:** Anteriormente, en el capítulo 4, sección 4.6, página 79, se han diseñado dos algoritmos basados en técnicas deterministas para resolver el juego.

- ✓ **Implementación de algoritmos no deterministas para la resolución del puzle:** En el capítulo 4, sección 4.7, página 84, también se ha generado un algoritmo capaz de resolver el juego del "2048" utilizando estrategias no deterministas.

- ✓ **Resultados estadísticos:** La interfaz anteriormente mencionada recoge todas las estadísticas necesarias por cada prueba que realiza, lo que resultó muy útil a la hora de realizar el capítulo 5 de la presente memoria. Como se puede apreciar, los resultados obtenidos son muy superiores a los pretendidos.

- ✓ **Resolución del juego estándar:** Como se vio en el capítulo anterior, varias configuraciones alcanzaron puntuaciones superiores a los 20.000 puntos por los seres humanos.

- ✓ **Conclusiones:** A lo largo de todo el proyecto se han extraído los resultados y las decisiones tomadas. El resultado generalizado ha sido muy satisfactorio, siendo posible la resolución exitosa de todos los objetivos planteados en el capítulo 3, página 39.

- ✓ **Planificación de todo el proyecto:** Como se podrá ver en el capítulo 7, página 123, de esta memoria, se ha realizado un seguimiento profundo de los costes, la ejecución de las tareas y el tiempo utilizado en las mismas.

- ✓ **Resumen en otro idioma:** En el anexo V, página 158 se podrá encontrar un resumen de todos los pasos realizados a lo largo de este proyecto en inglés.

6.2. Conclusiones finales

Aunque ha requerido mucho esfuerzo, se han conseguido alcanzar todos los objetivos propuestos de una manera muy satisfactoria.

Los problemas más importantes que se han encontrado a lo largo de la realización de este trabajo han sido el aprendizaje y uso del lenguaje de programación *Python*, así como de la librería *Wxpython*. Al no haber utilizado nunca estas herramientas, se cometían inicialmente muchos errores a la hora de programar el proyecto. Sin embargo, al ir adquiriendo conocimientos sobre las mismas, el número de errores se fue reduciendo de forma paulatina hasta desaparecer.

De igual manera, he aprendido el uso e implementación de potentes algoritmos de Inteligencia Artificial como el *Minimax*, el α-β y Monte Carlo, que se utilizan a menudo en cualquier ámbito de esta fascinante disciplina.

Con respecto a la motivación personal con la que se abordaba este proyecto, se ha conseguido satisfacer la curiosidad de si la estrategia seguida era la adecuada, debido a que la función de evaluación que ha obtenido mejores resultados que sus alternativas es la que usualmente utilizo. Además, el programa obtiene con frecuencia mejores puntuaciones que yo, lo que también satisface enormemente los objetivos y la motivación que han llevado a realizar este proyecto.

6.3. Líneas futuras:

A continuación, se describirán una serie de mejoras por las que se puede continuar para mejorar este proyecto:

6.3.1. Código en otro lenguaje de programación

Python es un lenguaje de programación muy sencillo de aprender y de usar. Sin embargo, no tiene la fuerza computacional de otros lenguajes, como puede ser C++. Es por esta razón que una de las posibles mejoras sería exportar el código de los algoritmos a este lenguaje, e integrarlo con la interfaz ya generada.

De este modo, seguramente se aligeraría en gran medida la carga computacional del problema, por lo que se podrían hacer pruebas más grandes, ya sea por número de pruebas o por profundidad.

6.3.2. Estudio de otros tableros (5x5, 3x3, tridimensional)

En este proyecto sólo se ha tenido en cuenta el tablero clásico de 4x4. Sin embargo, también podría ser interesante cambiar el tamaño para observar el comportamiento del programa con diferentes tamaños.

Debido al cambio de tamaño, el objetivo final también deberá ser distinto. En vez de intentar conseguir una pieza de 2048 en un tablero de 3x3 (que resultaría imposible), se tendría que poner como objetivo conseguir piezas de valor 128. De manera similar, en un tablero de 5x5 conseguir 2048 es muy sencillo debido al enorme espacio, por lo tanto, el verdadero reto en este caso sería conseguir fichas de 65536 o superior.

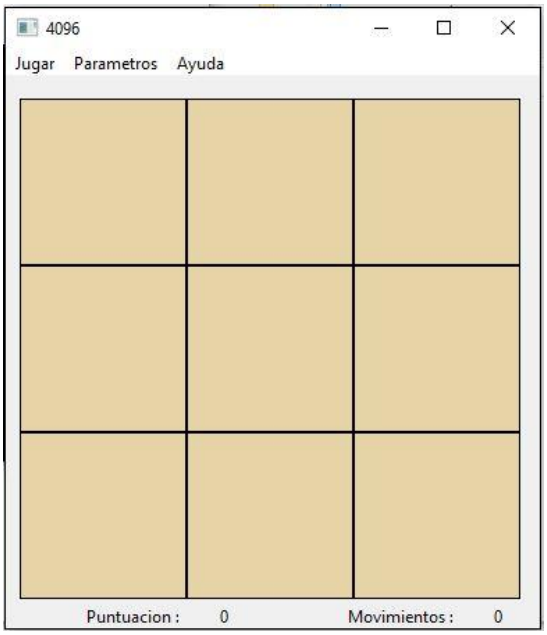


Ilustración 61 Ejemplo tablero 3x3

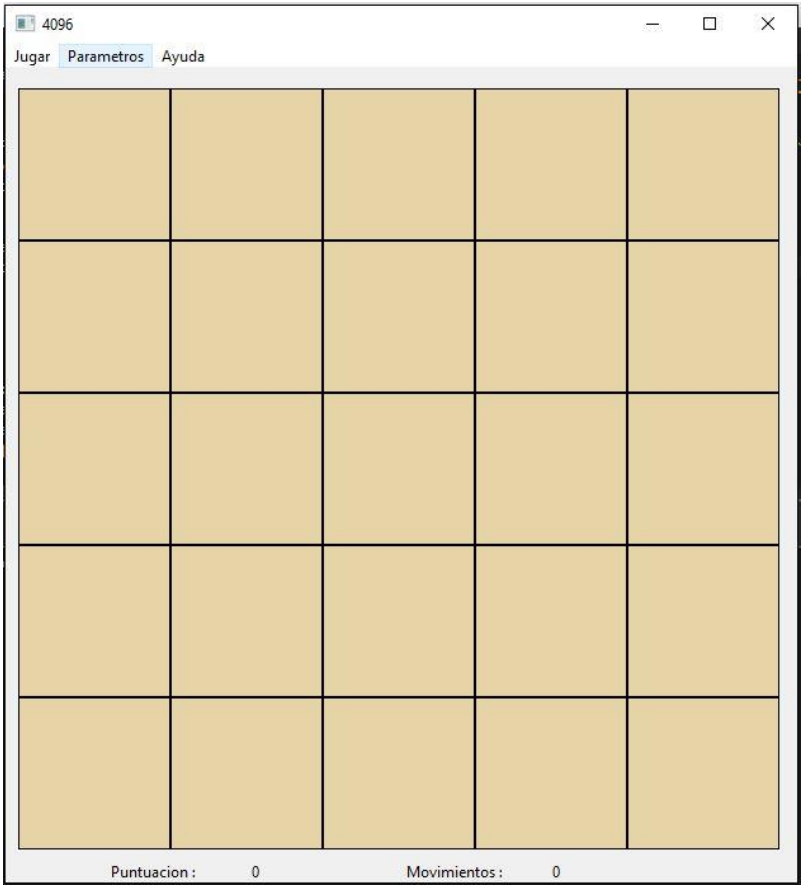


Ilustración 62 Ejemplo de tablero 5x5

También podría hacerse un estudio no sólo de cambio de tamaño, sino también de movimientos. Una posible idea sería tener dos tableros de 3x3 superpuestos, de tal manera que además de los movimientos tradicionales se añadieran hacia arriba y hacía debajo. El objetivo en este caso sería también de 2048 debido a la cercanía de las posibilidades; 18 en este caso y 16 en el del tablero clásico

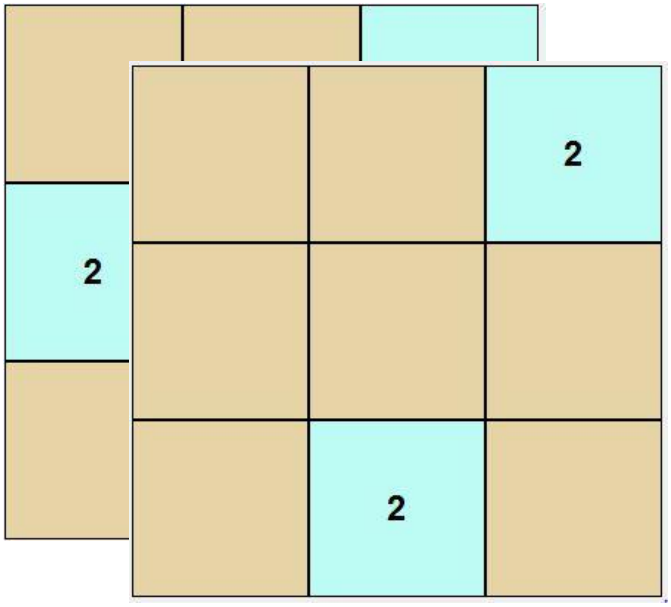


Ilustración 63 Tablero tridimensional

6.3.3. Búsqueda híbrida

Como se ha ido explicando anteriormente, (capítulo 4, sección 4.6, página 79) se han realizado dos tipos de búsquedas a lo largo de este proyecto. La primera de ellas era muy pesimista, buscando la posibilidad menos arriesgada de todas las que dispone. La segunda, más optimista, era capaz de correr ciertos riesgos con tal de obtener mejores resultados.

Como se puede prever, al no funcionar de la misma forma, cada una de ellas obtendrá distintos resultados para el mismo estado, dependiendo estrictamente del número de casillas libres. Cuantas más casillas libres haya, más viable es la toma de riesgos puesto que la tasa del mismo es baja. Sin embargo, en caso contrario funcionaría mejor la búsqueda sin riesgo puesto que la posibilidad de que el azar coloque una ficha en un mal lugar es muy alta.

Con estos conceptos, se pensó en realizar la llamada “búsqueda híbrida”. Simplemente consistiría en marcar un límite entre ambas búsquedas. Si el número de casillas libres es superior a esa marca, se haría búsqueda optimista. En caso contrario, se realizaría la búsqueda sin riesgos.

El único problema es encontrar qué valor de casillas vacías es el óptimo. Para encontrarlo, sería necesario un gran número de pruebas, con su comparación y su justificación. Es por ello que se ha decidido dejar esta mejora para un futuro.

6.3.4. Búsqueda optimista

De forma análoga a la mejora anterior, y como también se explicó en el capítulo 4, sección 4.4.2, página 47, se descartó hacer pruebas con búsqueda optimista debido a que se teorizó que los riesgos asumidos serían demasiado grandes como para mejorar los resultados frente a las búsquedas realista y pesimista. Sin embargo, para comprobar esta hipótesis se debería hacer una serie de pruebas que confirmen o refuten la teoría.

6.3.5. Mejora de reglas

Como se dijo en su momento, la resolución por medio de reglas requiere mucho esfuerzo por el simple hecho de generarlas.

Para resolver este tipo de problemas con reglas se tienen dos posibilidades.

Por un lado, un humano puede generarlas. Este procedimiento es muy costoso debido a la dificultad del mismo, ya que el manejo de excepciones a medida que el número de reglas aumenta suele ser muy grande. Es por esta razón que, si se decidiera mejorar las reglas para resolver el juego, no se seleccionaría este método.

La otra posibilidad sería el de la generación de reglas por medio de aprendizaje automático. Se ha demostrado que un ordenador utilizando estas técnicas puede crear reglas muy fuertes que rivalizarían con las creadas por un humano. Estas reglas se basan en conclusiones sacadas por miles y miles de partidas, extrayendo la información y reutilizándola para su posterior uso.

De tener la posibilidad, sería un buen ejercicio intentar resolver el juego del 2048 por medio de técnicas de aprendizaje automático, pero debido a la enorme cantidad de datos necesaria no ha sido posible realizarlas en esta ocasión

6.3.6. Nuevas posibilidades de azar

En el juego clásico de 2048, el azar tiene un pequeño porcentaje de colocar una ficha 4 en lugar de una ficha 2. A la hora de programar esta característica en el juego, no supone ninguna dificultad. Sin embargo, a la hora de tratar con la aleatoriedad en los algoritmos de búsqueda, el problema crece. Esto se debe a que las posibilidades se han multiplicado por dos, con lo que la carga computacional también crece de manera muy grande.

En el futuro, se podrían realizar modificaciones del código para que tengan en cuenta esta posibilidad, pero este cambio no es prioritario puesto que con total seguridad la resolución del mismo sería muchísimo más costosa.

6.3.7. Estudio con variación de tiempo del algoritmo de Monte Carlo

Como se explicó en su momento, el criterio de parada del algoritmo de Monte Carlo es un intervalo de tiempo: pasado un segundo de ejecución del método, el programa tomará una decisión en función de las estadísticas recogidas.

En este proyecto, solamente se ha probado un intervalo de tiempo. Podría ser interesante observar cómo afectaría a las jugadas el incrementar o disminuir este periodo, con el fin de obtener el valor óptimo para este parámetro.

6.3.8. Definición y pruebas de una nueva función de evaluación enfocada al algoritmo de Monte Carlo

Con la ejecución del algoritmo de Monte Carlo, daba la sensación de que la función de evaluación utilizada no explotaba con eficiencia el potencial de este método. Por lo tanto, la mejor opción sería diseñar y probar una serie de nuevas funciones de evaluación totalmente enfocadas para optimizar el uso de este algoritmo en este problema en concreto. Una vez diseñadas, se tendría que hacer un estudio intensivo para comprobar su validez.

6.3.9. Mejora de la interfaz gráfica

La interfaz realizada es una herramienta muy simple que permitía realizar pruebas a nivel de diseñador. Sin embargo, si se quisiera distribuir este software, sería necesaria la depuración y mejora de la interfaz gráfica en aras de mejorar su utilidad y el uso intuitivo por parte del usuario.

6.3.10. Adaptación para dispositivos móviles

Siendo un juego que nació en la plataforma de dispositivos móviles, la adaptación del programa para que pueda ser utilizada por usuarios de dispositivos móviles podría ser otra posible mejora.

6.3.11. Uso de otras técnicas de Inteligencia Artificial

La última mejora sería la implementación y la prueba de otros algoritmos que utilicen otras técnicas de Inteligencia Artificial y compararlas con el resultado obtenido en este proyecto. Se podría utilizar Redes de Neuronas, Algoritmos Genéticos y Evolutivos y Aprendizaje Automático.

Capítulo 7: Planificación y presupuesto

A continuación, se mostrará la planificación realizada para la realización del proyecto, necesaria para completar el objetivo número 13 especificado en la página 39.

7.1. Planificación inicial

Se comenzará con la planificación de las tareas del proyecto. Se decidió que fueran las siguientes:

Planificación

	Tarea	Descripción
Etapa 1	Pasos previos	
Fase 1.1	Estudio del lenguaje Python	Estudio y asimilación de los conceptos básicos del lenguaje base.
Fase 1.2	Estudio de los algoritmos utilizados	Estudio y comprensión de los conceptos básicos de los algoritmos originales que se pretenden usar en este proyecto.
Fase 1.3	Estudio de la complejidad del problema	Estudio y comprensión la complejidad y dificultad del puzle desde el punto de vista computacional.
Fase 1.4	Extracción de requisitos	Especificación de los requisitos que satisfagan las necesidades impuestas por los objetivos.
Fase 1.5	Planificación y presupuesto estimado	Realización del presupuesto y la planificación iniciales de este proyecto.
Etapa 2	Diseño	
Fase 2.1	Adecuación de los algoritmos utilizados al problema en cuestión	Implementación de los algoritmos que serán utilizados para la comprobación de las hipótesis planteadas en este proyecto.
Fase 2.2	Diseño arquitectónico de la aplicación	Diseño del sistema que se utilizará para la implementación del programa.
Etapa 3	Programación	
Fase 3.1	Programación de la interfaz	Realización de la interfaz gráfica del programa.
Fase 3.2	Programación del agente	Implementación de los algoritmos necesarios para la ejecución de las pruebas.
Fase 3.3	Conexión entre interfaz y agente	Unión entre la interfaz gráfica y el agente de Inteligencia Artificial.
Etapa 4	Pruebas	
Fase 4.1	Planificación de las pruebas	Preparación del script que ejecutará las pruebas de forma automática.
Fase 4.2	Realización de las pruebas	Ejecución y toma de resultados de las pruebas.
Etapa 5	Documentación	
Fase 5.1	Escritura del documento	Escritura del presente documento.
Fase 5.2	Revisión del documento	Revisión del presente documento.

Tabla 75 Descripción de las tareas

7.1.1. Tabla de planificación inicial

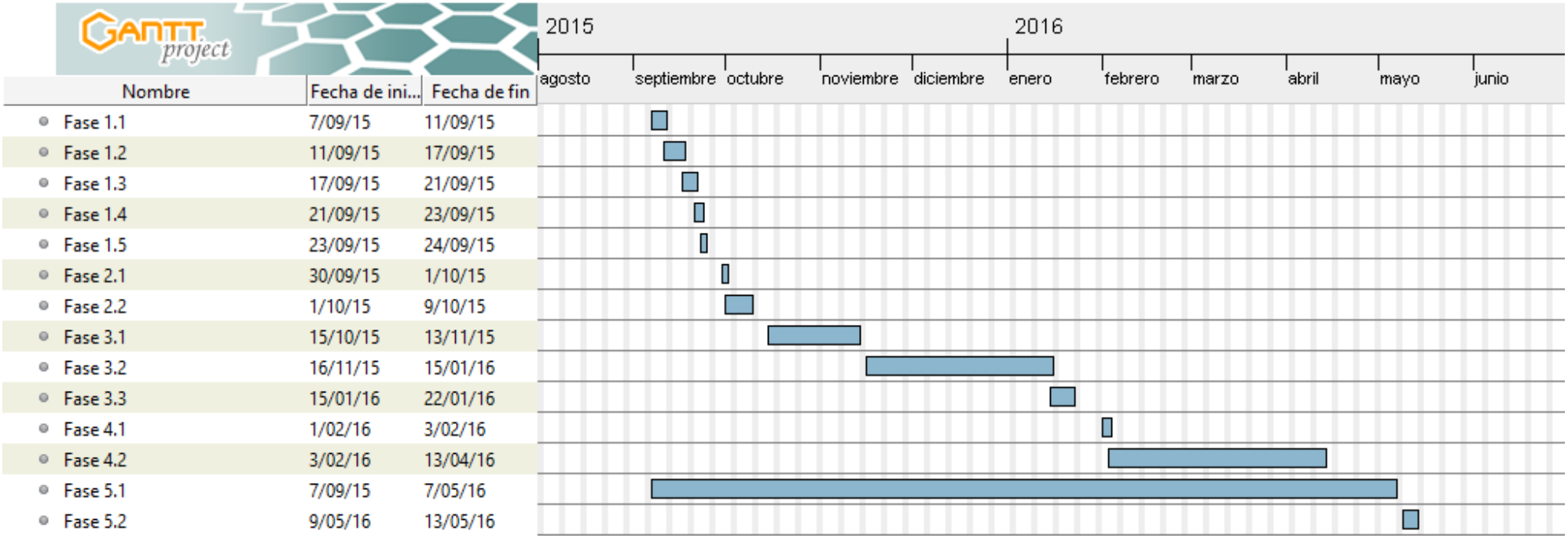
La planificación inicial del proyecto se planteó de la siguiente manera:

Planificación				
	Tarea	Tiempo estimado	Inicio	Fin
Etapa 1	Pasos previos			
Fase 1.1	Estudio del lenguaje Python	5 días	07/09/15	11/09/15
Fase 1.2	Estudio de los algoritmos utilizados	6 días	11/09/15	17/09/15
Fase 1.3	Estudio de la complejidad del problema	4 días	17/09/15	21/09/15
Fase 1.4	Extracción de requisitos	2 días	21/09/15	23/09/15
Fase 1.5	Planificación y presupuesto estimado	1 día	23/09/15	24/09/15
Etapa 2	Diseño			
Fase 2.1	Adecuación de los algoritmos utilizados al problema en cuestión	1 día	30/09/15	01/10/15
Fase 2.2	Diseño arquitectónico de la aplicación	10 días	01/10/15	11/10/15
Etapa 3	Programación			
Fase 3.1	Programación de la interfaz	1 mes	15/10/15	15/11/15
Fase 3.2	Programación del agente	2 meses	16/11/15	15/01/16
Fase 3.3	Conexión entre interfaz y agente	7 días	15/01/16	22/01/16
Etapa 4	Pruebas			
Fase 4.1	Planificación de las pruebas	2 días	01/02/16	03/02/16
Fase 4.2	Realización de las pruebas	2 meses, 10 días	03/02/16	13/04/16
Etapa 5	Documentación			
Fase 5.1	Escritura del documento	9 meses	07/09/15	07/05/16
Fase 5.2	Revisión del documento	4 días	09/05/16	13/05/16

Tabla 76 Planificación del proyecto inicial

7.1.1. Diagrama de Gantt inicial

El siguiente diagrama de Gantt resume los tiempos que se necesitarán para cada una de las tareas:



Nombre	Fecha de ini...	Fecha de fin
Fase 1.1	7/09/15	11/09/15
Fase 1.2	11/09/15	17/09/15
Fase 1.3	17/09/15	21/09/15
Fase 1.4	21/09/15	23/09/15
Fase 1.5	23/09/15	24/09/15
Fase 2.1	30/09/15	1/10/15
Fase 2.2	1/10/15	9/10/15
Fase 3.1	15/10/15	13/11/15
Fase 3.2	16/11/15	15/01/16
Fase 3.3	15/01/16	22/01/16
Fase 4.1	1/02/16	3/02/16
Fase 4.2	3/02/16	13/04/16
Fase 5.1	7/09/15	7/05/16
Fase 5.2	9/05/16	13/05/16

Ilustración 64 Diagrama de Gannt inicial

7.2. Planificación real

7.2.1. Tabla de planificación real

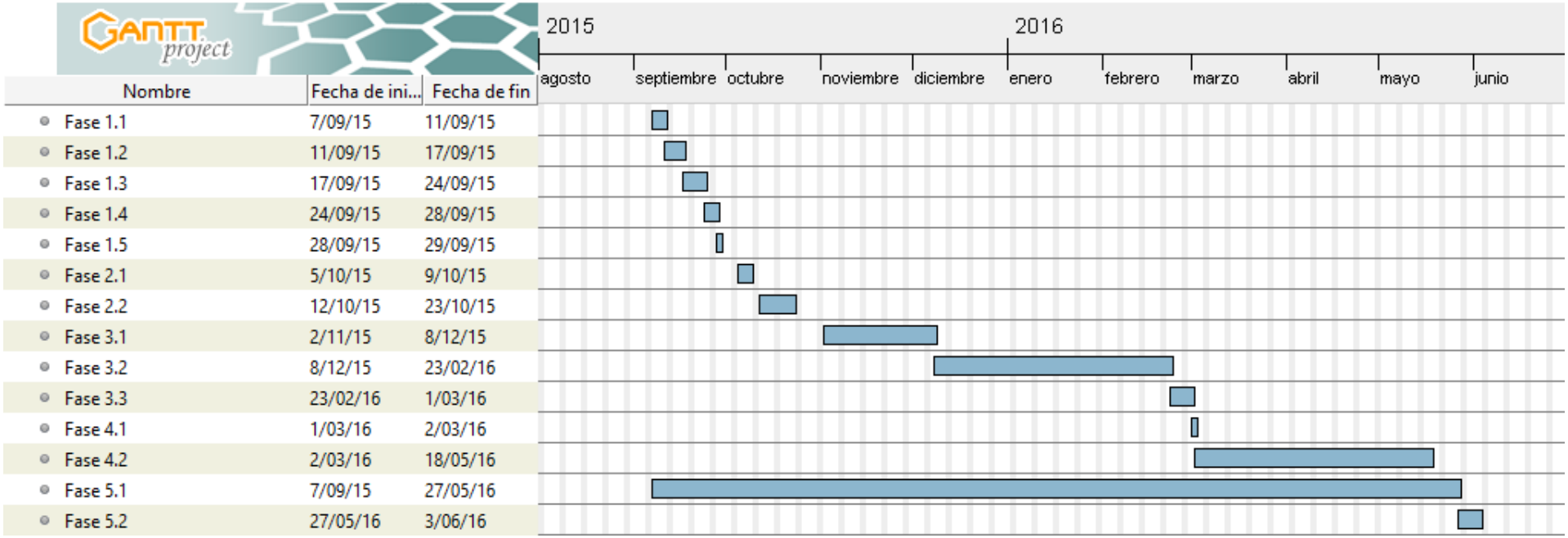
Ahora se procederá a mostrar los tiempos reales empleados en cada una de las tareas. Como se puede apreciar, hay un retraso general en cuanto a la planificación inicial, si bien esta demora es más notable en la realización de las pruebas.

Planificación				
	Tarea	Tiempo real	Inicio	Fin
Etapa 1	**Pasos previos**			
Fase 1.1	Estudio del lenguaje Python	2 días	07/09/15	11/09/15
Fase 1.2	Estudio de los algoritmos utilizados	6 días	11/09/15	17/09/15
Fase 1.3	Estudio de la complejidad del problema	7 días	17/09/15	24/09/15
Fase 1.4	Extracción de requisitos	4 días	24/09/15	28/09/15
Fase 1.5	Planificación y presupuesto estimado	1 día	28/09/15	29/09/15
Etapa 2	**Diseño**			
Fase 2.1	Adecuación de los algoritmos utilizados al problema en cuestión	7 días	03/10/15	10/10/15
Fase 2.2	Diseño arquitectónico de la aplicación	15 días	10/10/15	25/10/15
Etapa 3	**Programación**			
Fase 3.1	Programación de la interfaz	1 mes y 7 días	01/11/15	08/12/15
Fase 3.2	Programación del agente	2 meses y 15 días	08/12/15	23/02/16
Fase 3.3	Conexión entre interfaz y agente	7 días	23/02/16	30/02/16
Etapa 4	**Pruebas**			
Fase 4.1	Planificación de las pruebas	2 días	01/03/16	03/03/16
Fase 4.2	Realización de las pruebas	3 meses y 15 días	03/02/16	18/05/16
Etapa 5	**Documentación**			
Fase 5.1	Escritura del documento	9 meses y 20 días	07/09/15	27/05/16
Fase 5.2	Revisión del documento	7 días	27/05/16	03/06/16

Tabla 77 Planificación del proyecto real

7.2.2. Diagrama de Gantt real

El diagrama correspondiente a la tabla anterior:



Nombre	Fecha de ini...	Fecha de fin
Fase 1.1	7/09/15	11/09/15
Fase 1.2	11/09/15	17/09/15
Fase 1.3	17/09/15	24/09/15
Fase 1.4	24/09/15	28/09/15
Fase 1.5	28/09/15	29/09/15
Fase 2.1	5/10/15	9/10/15
Fase 2.2	12/10/15	23/10/15
Fase 3.1	2/11/15	8/12/15
Fase 3.2	8/12/15	23/02/16
Fase 3.3	23/02/16	1/03/16
Fase 4.1	1/03/16	2/03/16
Fase 4.2	2/03/16	18/05/16
Fase 5.1	7/09/15	27/05/16
Fase 5.2	27/05/16	3/06/16

Ilustración 65 Diagrama de Gannt real

7.3. Comparación de las planificaciones

A continuación, se mostrará una tabla que contendrá la comparativa de las dos planificaciones, tanto la estimada como la real

Comparación de las estimaciones

	Tarea	Tiempo estimado	Tiempo real	Desfase
Etapa 1	**Pasos previos**			
Fase 1.1	Estudio del lenguaje Python	5 días	2 días	-3 días
Fase 1.2	Estudio de los algoritmos utilizados	6 días	6 días	-
Fase 1.3	Estudio de la complejidad del problema	4 días	7 días	3 días
Fase 1.4	Extracción de requisitos	2 días	4 días	2 días
Fase 1.5	Planificación y presupuesto estimado	1 día	1 día	-
Etapa 2	**Diseño**			
Fase 2.1	Adecuación de los algoritmos utilizados al problema en cuestión	1 día	7 días	6 días
Fase 2.2	Diseño arquitectónico de la aplicación	10 días	15 días	5 días
Etapa 3	**Programación**			
Fase 3.1	Programación de la interfaz	1 mes	1 mes y 7 días	7 días
Fase 3.2	Programación del agente	2 meses	2 meses y 15 días	15 días
Fase 3.3	Conexión entre interfaz y agente	7 días	7 días	-
Etapa 4	**Pruebas**			
Fase 4.1	Planificación de las pruebas	2 días	2 días	-
Fase 4.2	Realización de las pruebas	2 meses, 10 días	3 meses y 15 días	1 mes y 5 días
Etapa 5	**Documentación**			
Fase 5.1	Escritura del documento	9 meses	9 meses y 20 días	20 días
Fase 5.2	Revisión del documento	4 días	7 días	3 días
	Desfase total			**23 días**

Tabla 78 Comparación de las planificaciones

Se puede ver que ha habido un pequeño retraso generalizado en todas las tareas, pero se ha ido supliendo con los períodos de descanso y con la escritura del presente documento a lo largo de todo el proceso del proyecto. Sin embargo, las pruebas tardaron mucho más tiempo de lo previsto (más de un mes de desfase). Ese tiempo, una vez más, se empleó en la escritura de esta memoria, lo que permitió reducir el retraso en gran medida.

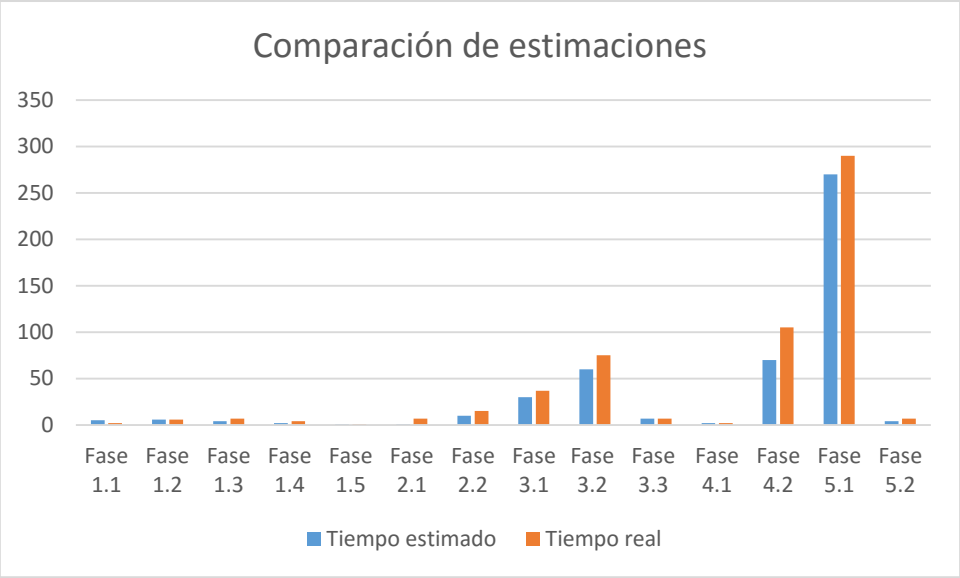


Ilustración 66 Comparación de estimaciones

Por todo ello, la planificación estimada no se pudo cumplir, pero únicamente se ha tenido un desfase de 23 días (del 13/05/16 al 03/06/16). También es destacable comentar que la planificación inicial era extremadamente optimista para poder tener tiempo de reacción ante los distintos problemas que se podrían producir y que, efectivamente, se produjeron.

7.4. Presupuesto

A continuación, se mostrará, de forma desglosada, el presupuesto del proyecto.

7.4.1. Coste de personal

Se comenzará con los costes de personal. Este proyecto se ha realizado por dos personas, por lo que el ingeniero informático aunará los roles de analista, diseñador, responsable de pruebas, programador y documentador.

Personal	Horas dedicadas	Salario/hora (€/hora)	Total (€)	Coste seguridad Social	Coste real
Jefe de proyecto	50	40	2000 €	472 €	2472 €
Ingeniero informático	1188	20 €	23.760 €	5.607,36 €	29.367,36 €
TOTAL			**25.760 €**	**6.079, 36 €**	**31.839,36 €**

Tabla 79 Costes de personal del proyecto

Las horas del ingeniero informático se han calculado a partir de los 297 días que ha durado el proyecto en total. El alumno le ha dedicado una media de 4 horas diarias, lo que se ha pagado a 20 € la hora. Además, se ha aplicado el 23,6% en concepto de la cuota empresarial a la Seguridad Social.

El coste de recursos humanos de este proyecto asciende a TREINTA Y UN MIL OCHOCIENTOS TREINTA Y NUEVE EUROS CON TREINTA Y SEIS CENTIMOS (31.839,36 €).

7.4.2. Hardware

A continuación, se mostrará los costes en Hardware del proyecto. El PC con el que se cuenta es el especificado en la Ilustración 33, página 89.

Recurso	Precio	Duración del proyecto	Amortización (años)	Precio/unidad(€)	Cantidad	Coste real
PC HP Pavilion 15nr	800 €	297 días	4 años	162,74 €	1	162,74 €
Impresora HP Deskjet F4580	75 €	297 días	4 años	10,94 €	1	10,94 €
TOTAL						**173,68 €**

Tabla 80 Costes de Hardware del proyecto

El total de costes de Hardware ha sido de CIENTO SETENTA Y TRES EUROS CON SESENTA Y OCHO CENTIMOS (173,68 €).

7.4.3. Software

En la siguiente tabla se podrá ver el dinero gastado en software en este proyecto.

Recurso	Precio	Duración del proyecto	Amortización (años)	Precio/unidad(€)	Cantidad	Coste real
Windows 7 professional	160 €	297 días	4 años	32,55 €	1	32,55 €
Microsoft Office 2013	99 €	297 días	4 años	20,14 €	1	20,14 €
Python	0 €	297 días	4 años	0 €	1	0 €
Wxpython						
Ninja-Ide	0 €	297 días	4 años	0 €	1	0 €
TOTAL						**52,69 €**

Tabla 81 Costes de software del proyecto

Tanto la licencia de Python como de la librería Wxpython, así como el IDE Ninja son gratuitas.

El coste de software asciende a CINCUENTA Y DOS EUROS CON SESENTA Y NUEVE CENTIMOS (52,69 €).

7.4.4. Material fungible

A continuación, se hará un inventario de todo el material de oficina necesario para la realización de este proyecto.

Recurso	Precio	Cantidad	Coste real
Pack cartucho tinta negra + color	34,99 €	4	139,96 €
Pack 500 folios DIN-A4	2,35 €	3	7,05 €
Pack de material de oficina	30 €	1	30 €
Bote de imprevistos	200 €	1	200 €
TOTAL			**377,01 €**

Tabla 82 Coste de material fungible del proyecto

El total ha sido de TRESCIENTOS SETENTA Y SIETE EUROS CON UN CENTIMO. (377,01 €).

7.4.5. Costes fijos

Por último, se han tenido en cuenta los costes fijos que han sido necesarios a lo largo de estos casi diez meses.

Recurso	Precio mensual	Duración del proyecto (meses aproximados)	Coste real
Internet	32 €	10	320 €
Luz	30 €	10	300 €
TOTAL			**620 €**

Tabla 83 Costes fijos del proyecto

Los costes fijos han sumado un total de SESCIENTOS VEINTE EUROS (620 €).

7.4.6. Coste total

El coste total se agrupa a continuación:

Recurso	Coste
Personal	31.839,36 €
Hardware	173,68 €
Software	52,69 €
Material fungible	377,01 €
Coste fijos	620 €
TOTAL	33.062,74 €

Tabla 84 Presupuesto del proyecto

En total, este proyecto ha costado TREINTA Y TRES MIL SESENTA Y DOS EUROS CON SETENTA Y CUATRO CENTIMOS (33.062,74 €). Como se trata de un proyecto fin de grado, los costes de personal pueden ser omitidos, lo que da un coste real de MIL DOSCIENTOS VEINTITRES EUROS CON TREINTA Y OCHO CENTIMOS (1.223,38 €)

Anexo I: Manual de usuario

Para este proyecto, se ha realizado un programa con una interfaz gráfica muy sencilla. Ésta es la susodicha interfaz:

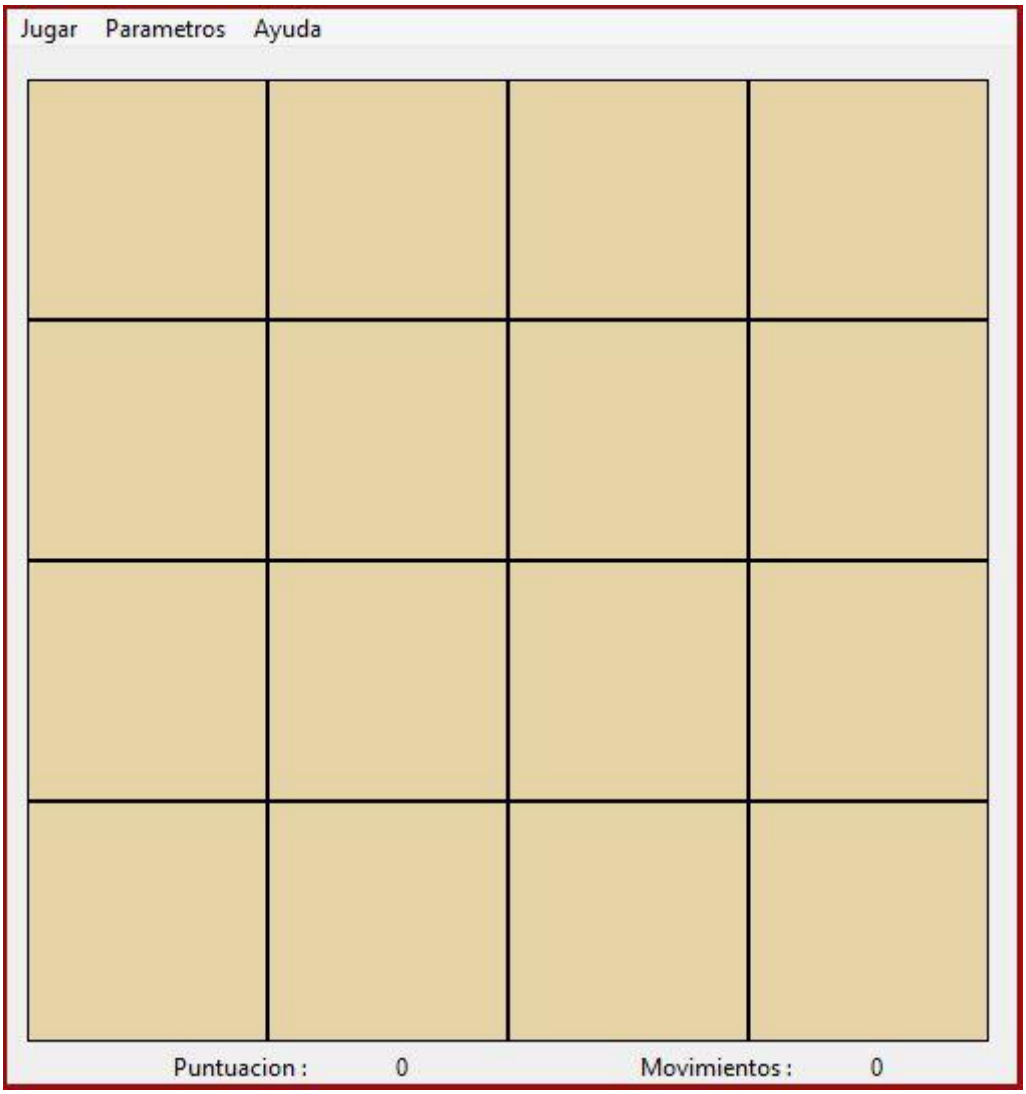


Ilustración 67 Interfaz gráfica

Como se puede apreciar, es una única ventana con un tablero clásico. En la parte inferior se muestran la puntuación y los movimientos totales de la partida en cuestión.

En el menú principal hay tres submenús, los cuales se explican a continuación.

El primero, llamado “Jugar”, permite al usuario utilizar el programa para probar el juego. Hay dos opciones:

- **Jugar manual:** Se empezará una partida nueva. Los comandos para la misma serán “w” o ↑ para hacer un movimiento arriba, “s” o ↓ para abajo, “a” o ← para izquierda y “d” o → para derecha. De igual forma, si se pulsa la tecla “i”, el programa ejecutará un movimiento en función de los parámetros indicados.

- **Jugar algoritmo:** Con esta opción, el programa empezará a jugar partidas de forma automática en función de los parámetros introducidos. Saldrá una ventana que indica el número de prueba que se está ejecutando en ese momento, así como el tiempo transcurrido, el tiempo estimado y el tiempo restante. Cuando acabe una partida, se pasarán los datos de la misma a un fichero de texto con un nombre que resumirá los parámetros introducidos para su referencia posterior.

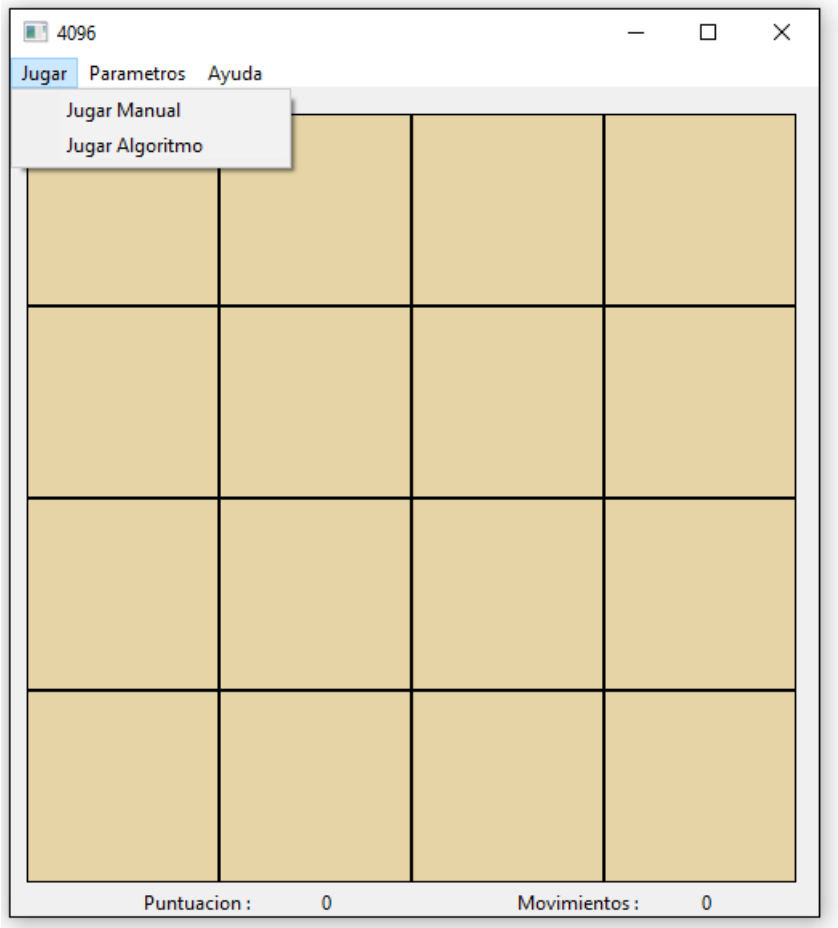


Ilustración 68 Menú jugar

El siguiente menú, “Parámetros”, permite cambiar los parámetros para ejecutar las pruebas. Se divide en varios submenús.

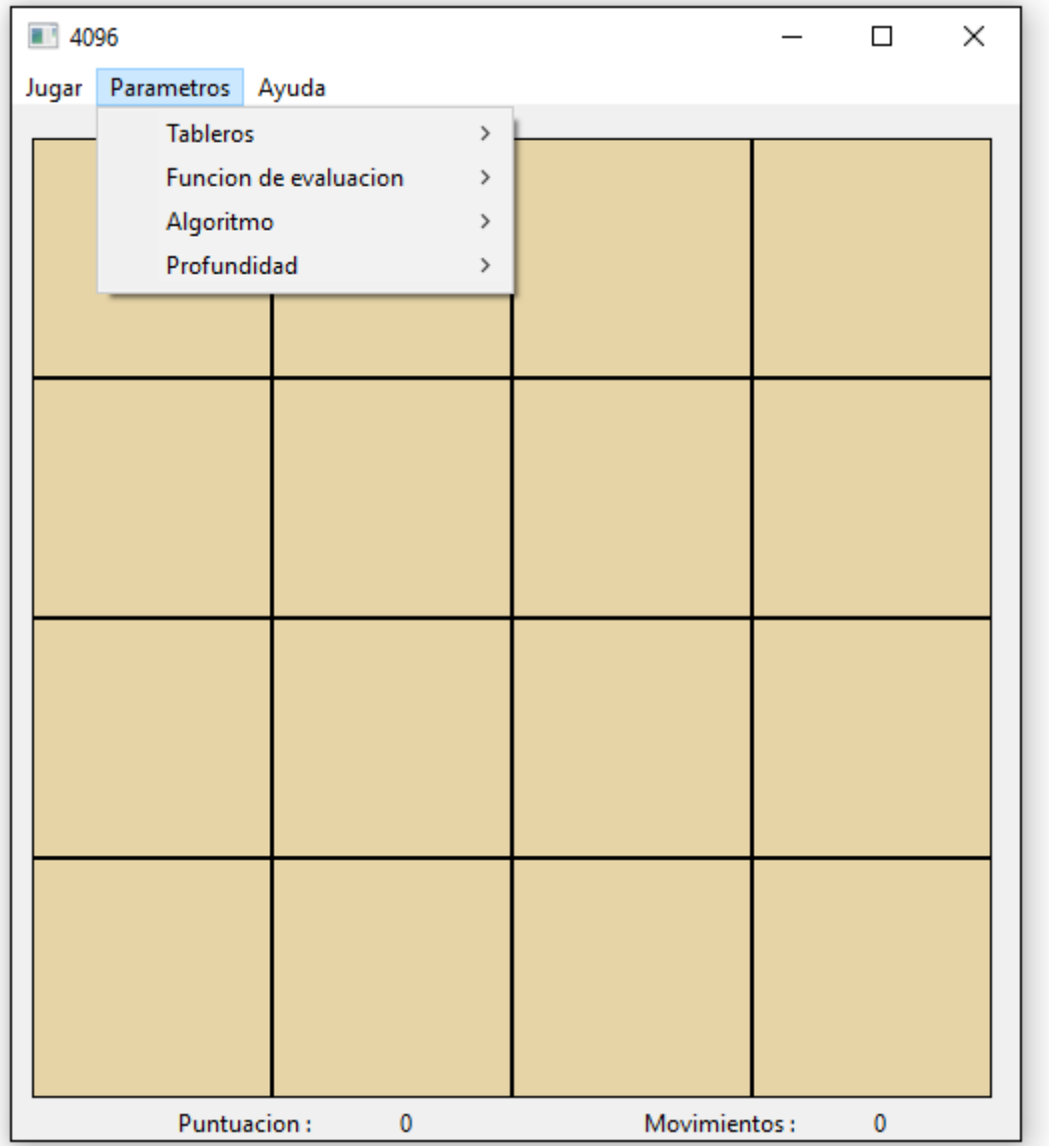


Ilustración 69 Menú parámetros

- **<u>Tableros:</u>** Aquí se indicará el tablero sobre el que se van a ejecutar las pruebas. Puede ser “3x3”, “4x4” o “5x5”.

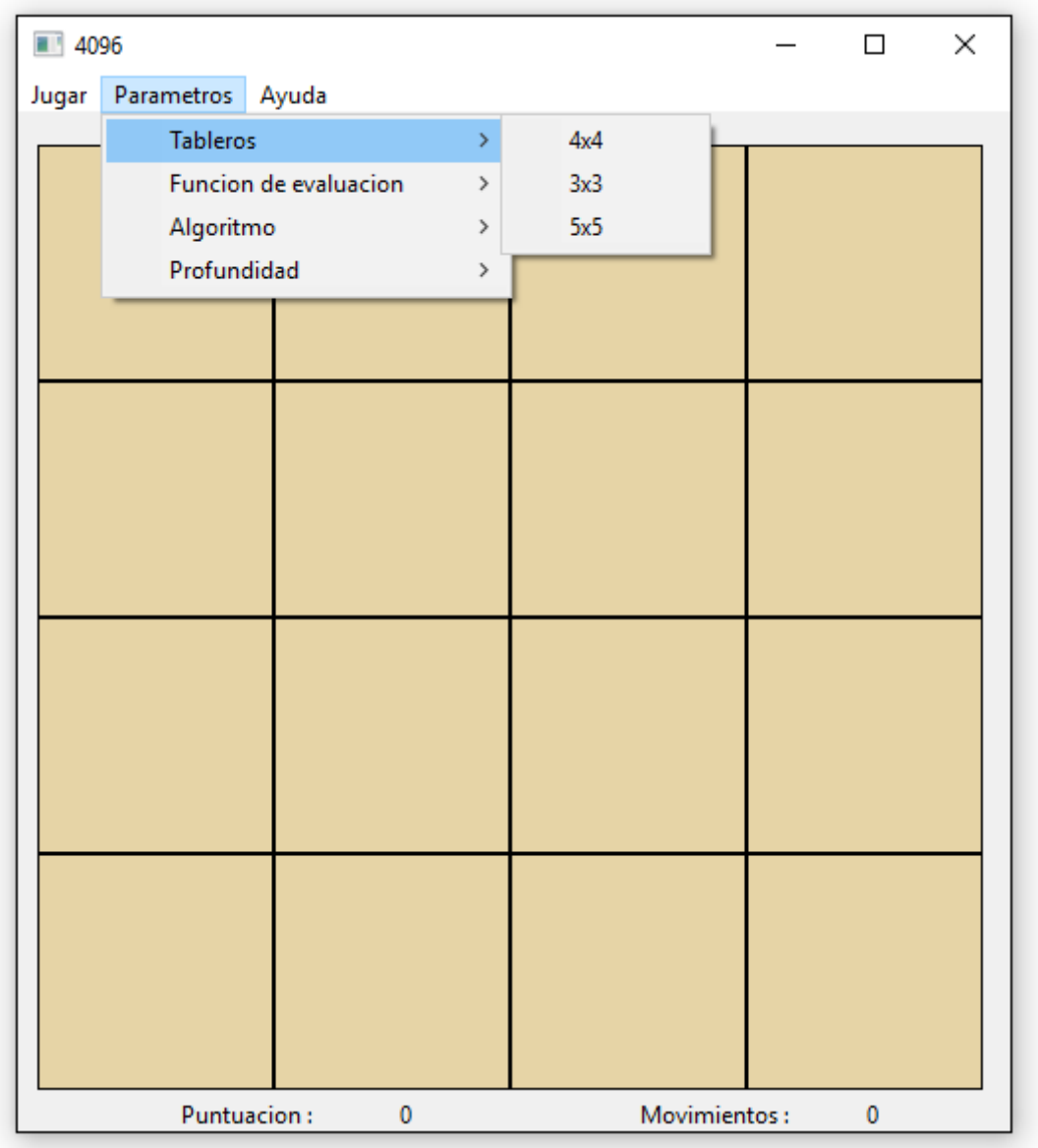


Ilustración 70 Menú tableros

- **<u>Función de evaluación:</u>** para seleccionar el modo de evaluación de los tableros. Se explica con detalle con más detenimiento en el Capítulo 2 de la memoria asociada a este proyecto.

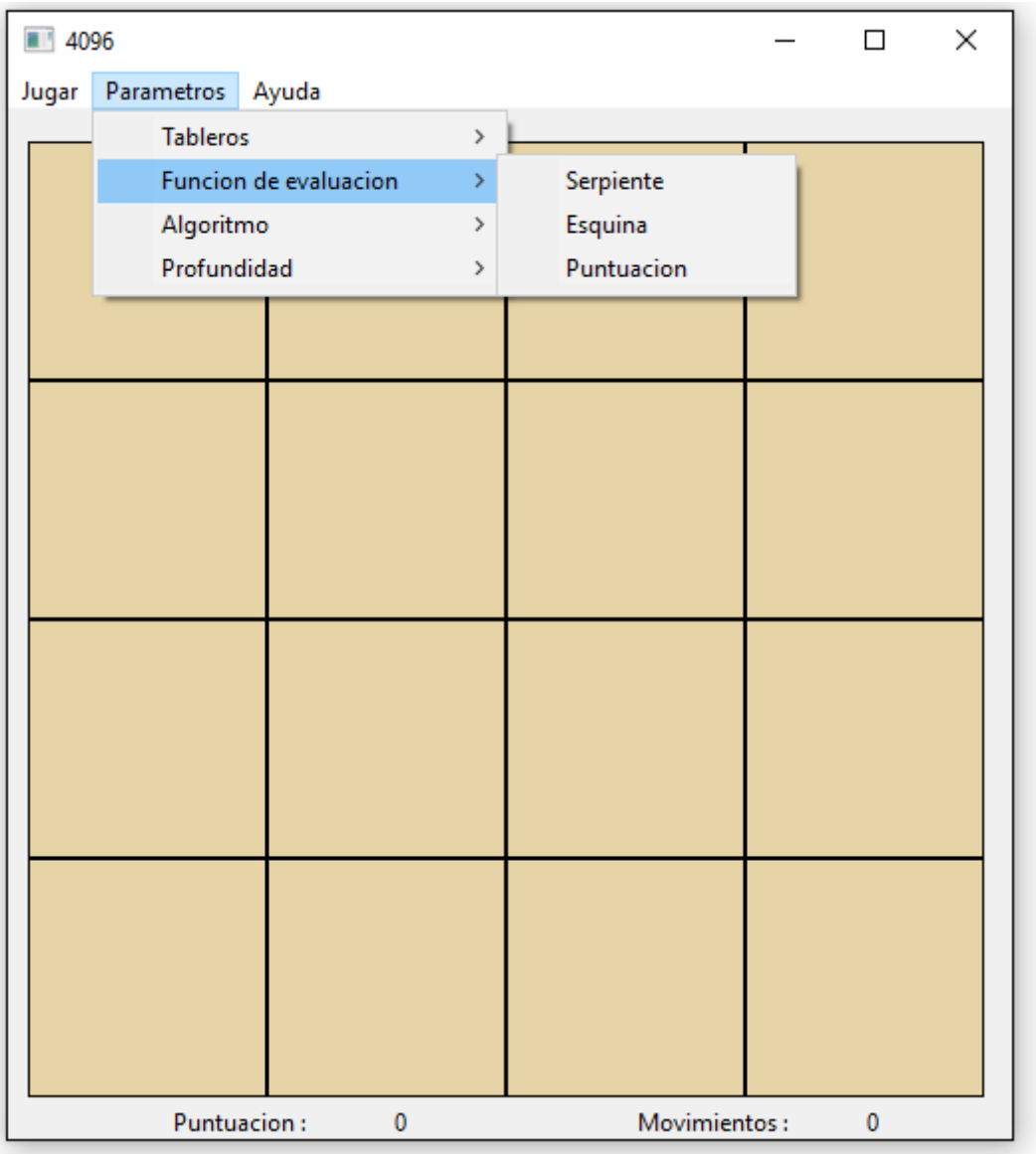


Ilustración 71 Menú función de evaluación

- **<u>Algoritmo:</u>** se utilizará para seleccionar el método de selección de sucesores. Se explica con más detenimiento en los capítulos 2 y 4 de la memoria asociada de proyecto.

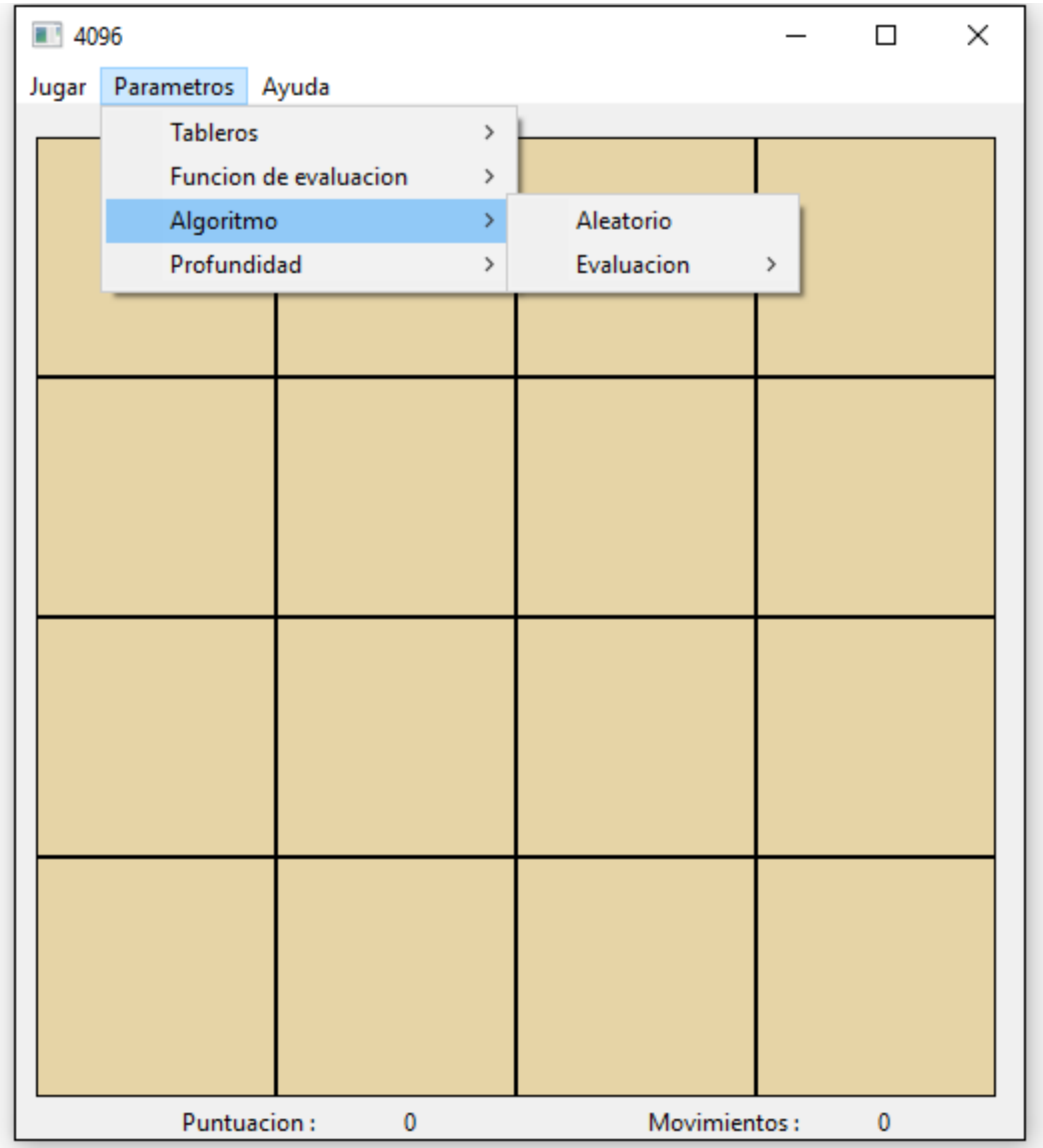


Ilustración 72 Menú algoritmo

- En caso de que se quiera seleccionar un algoritmo que no sea aleatorio, hay varias opciones:

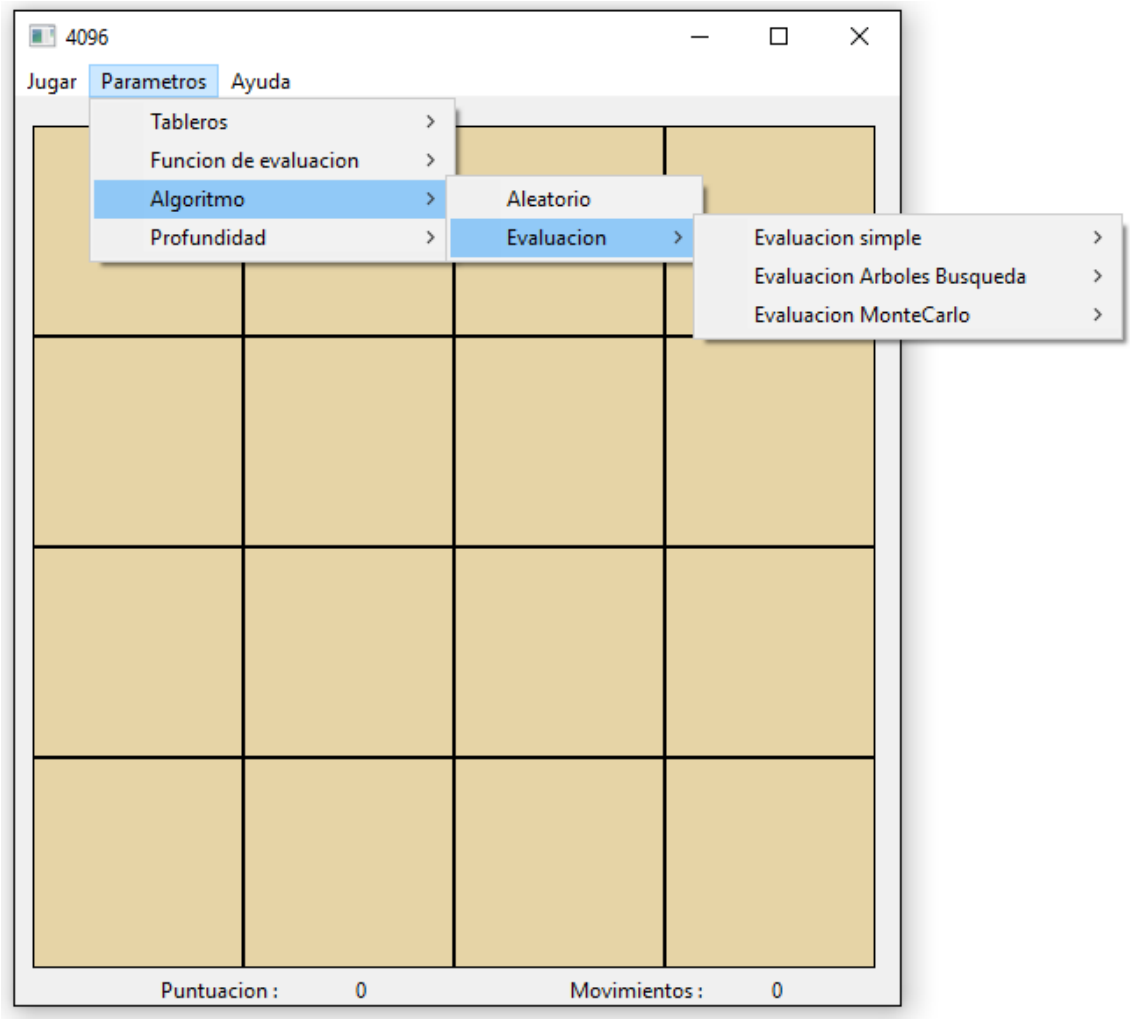


Ilustración 73 Menú algoritmo evaluación

- En caso de que la evaluación sea simple, hay estas opciones:

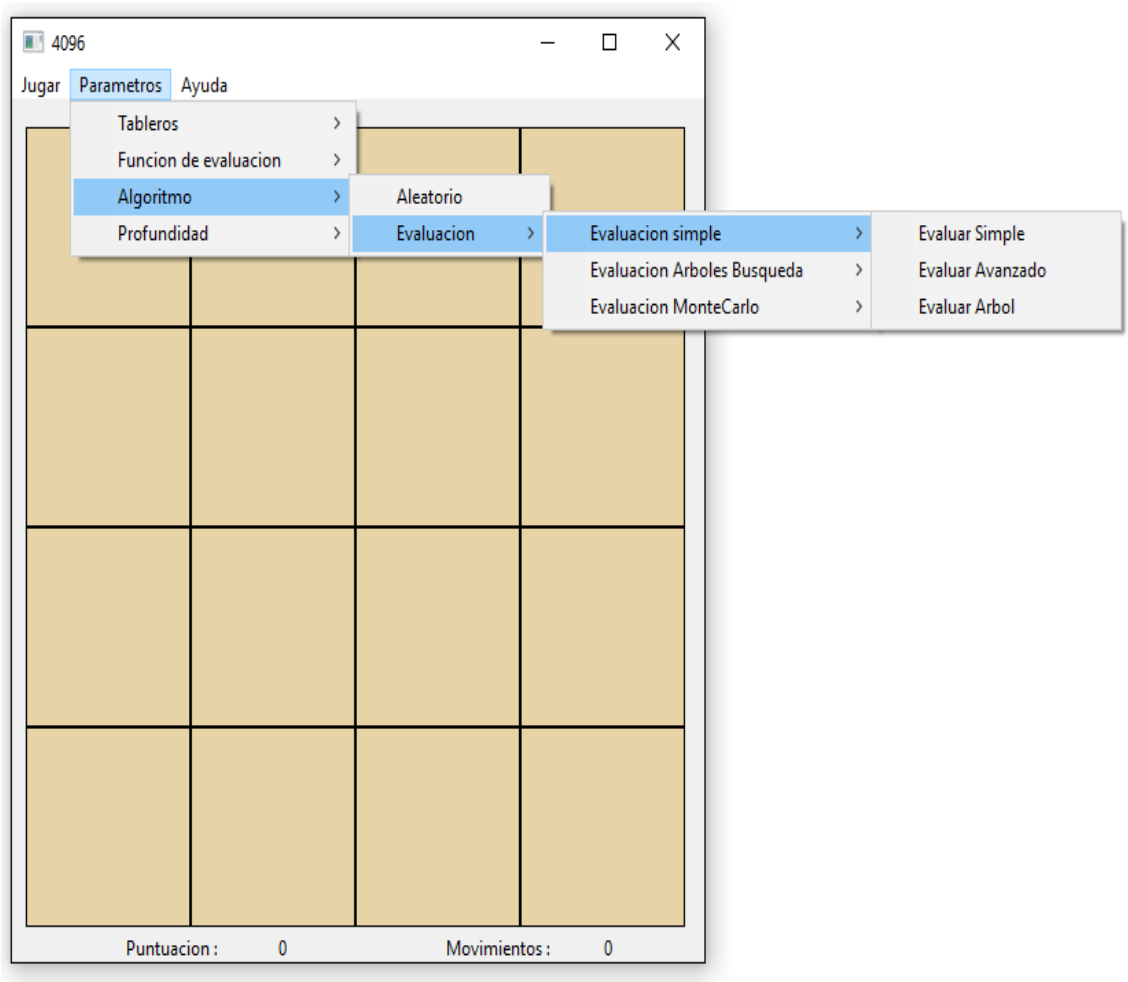


Ilustración 74 Menú evaluación simple

- En caso de que la evaluación sea un árbol de búsqueda, hay estas opciones:

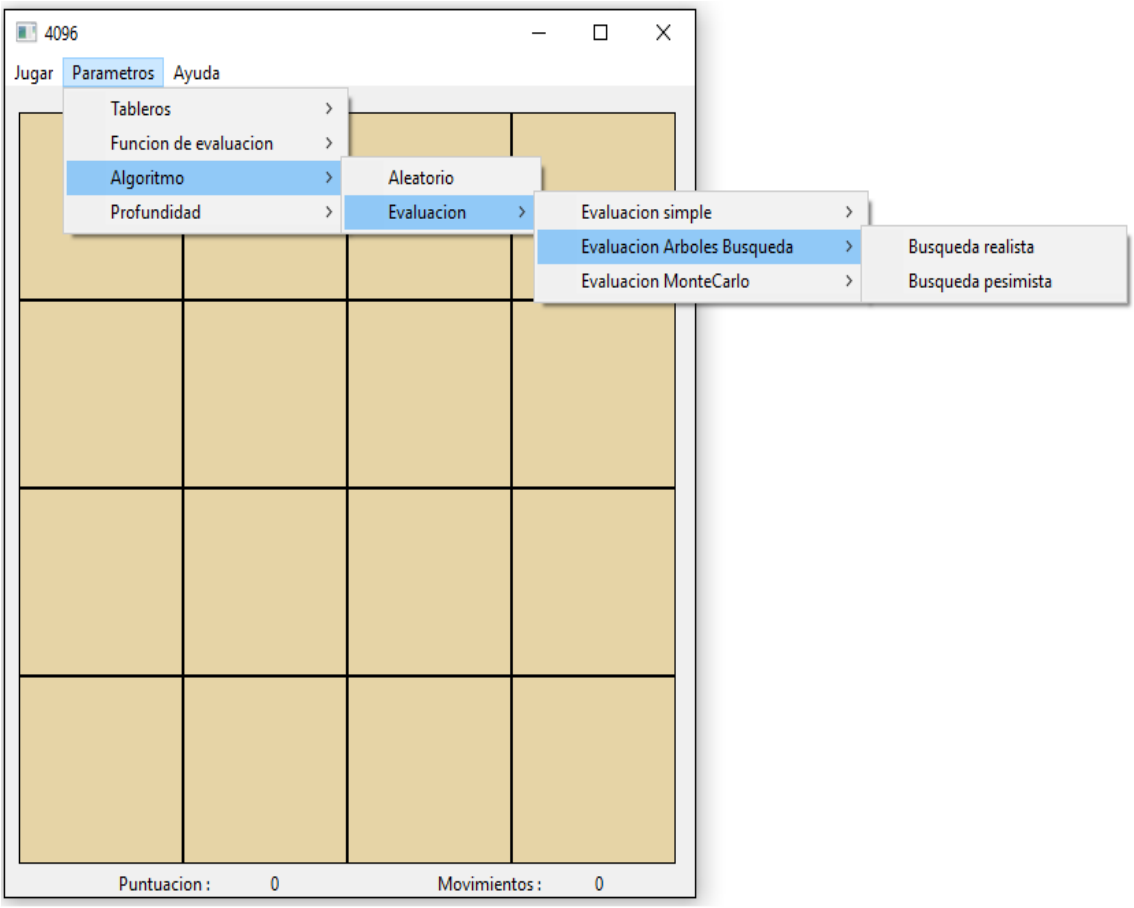


Ilustración 75 Menú evaluación árbol de búsqueda

- En caso de que la evaluación sea el algoritmo de Monte Carlo, hay estas opciones:

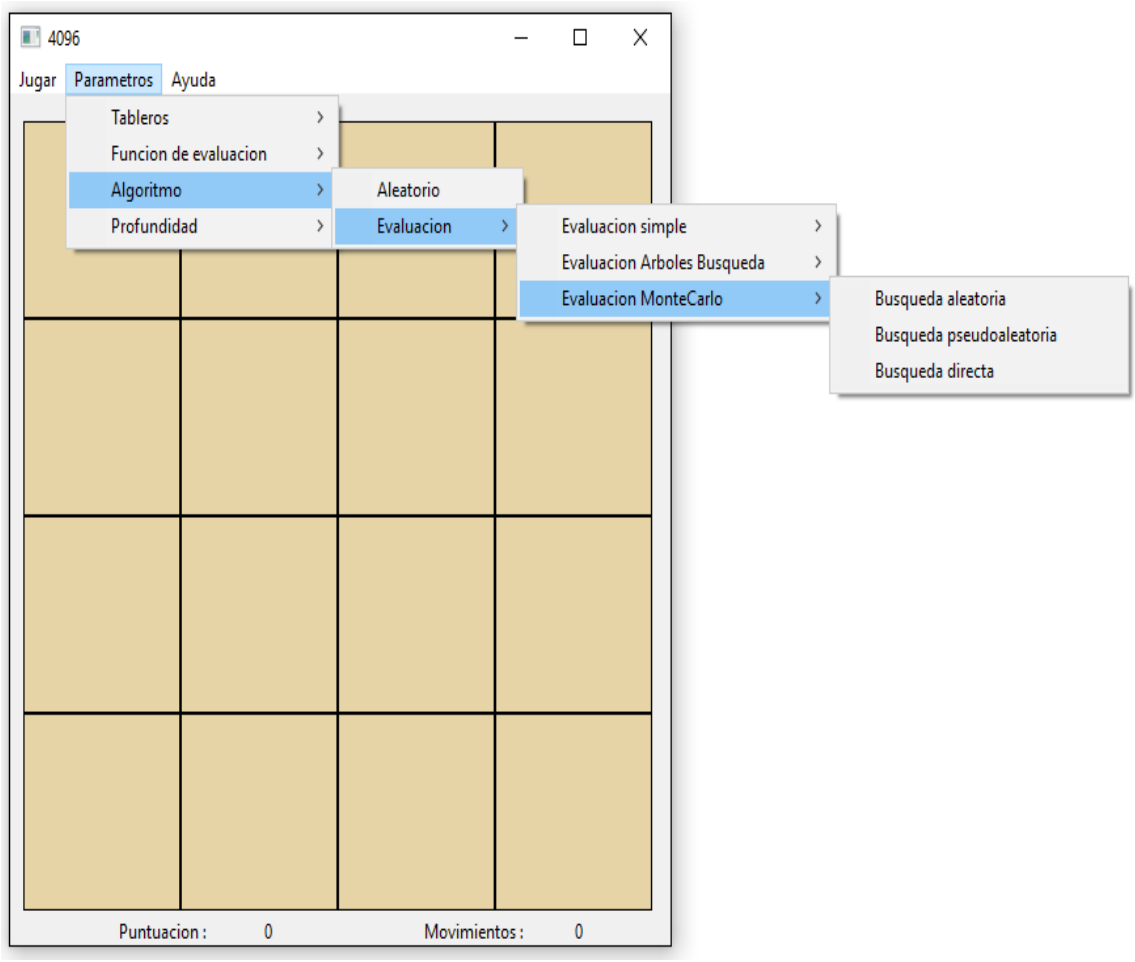


Ilustración 76 Menú evaluación algoritmo Monte Carlo

- **<u>Profundidad:</u>** existen algoritmos para los que es necesario seleccionar la profundidad en función del algoritmo indicado anteriormente.

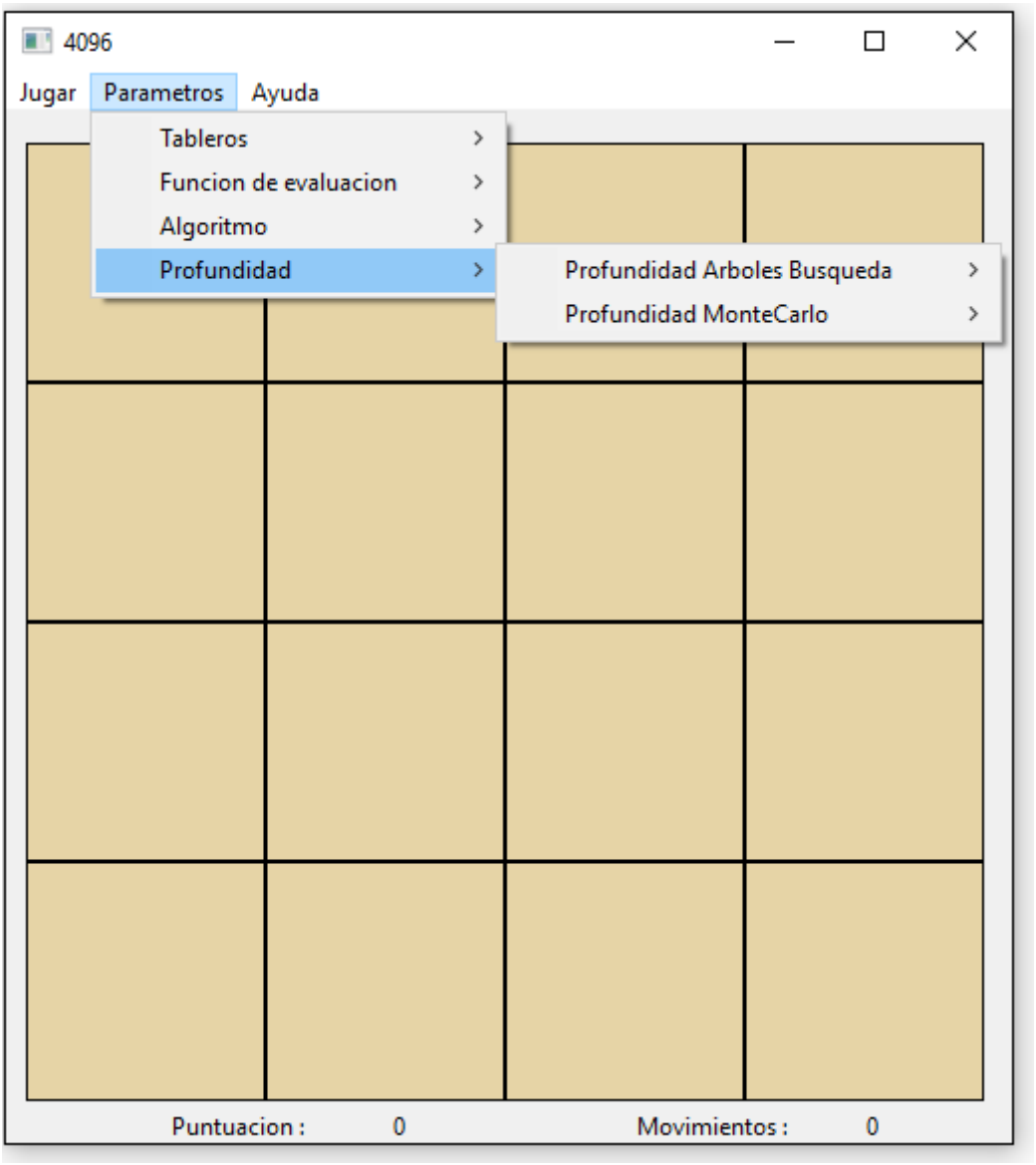


Ilustración 77 Menú profundidad

- En caso de que el algoritmo sea un árbol de búsqueda, las posibilidades son 4, 6, 8 y 10.

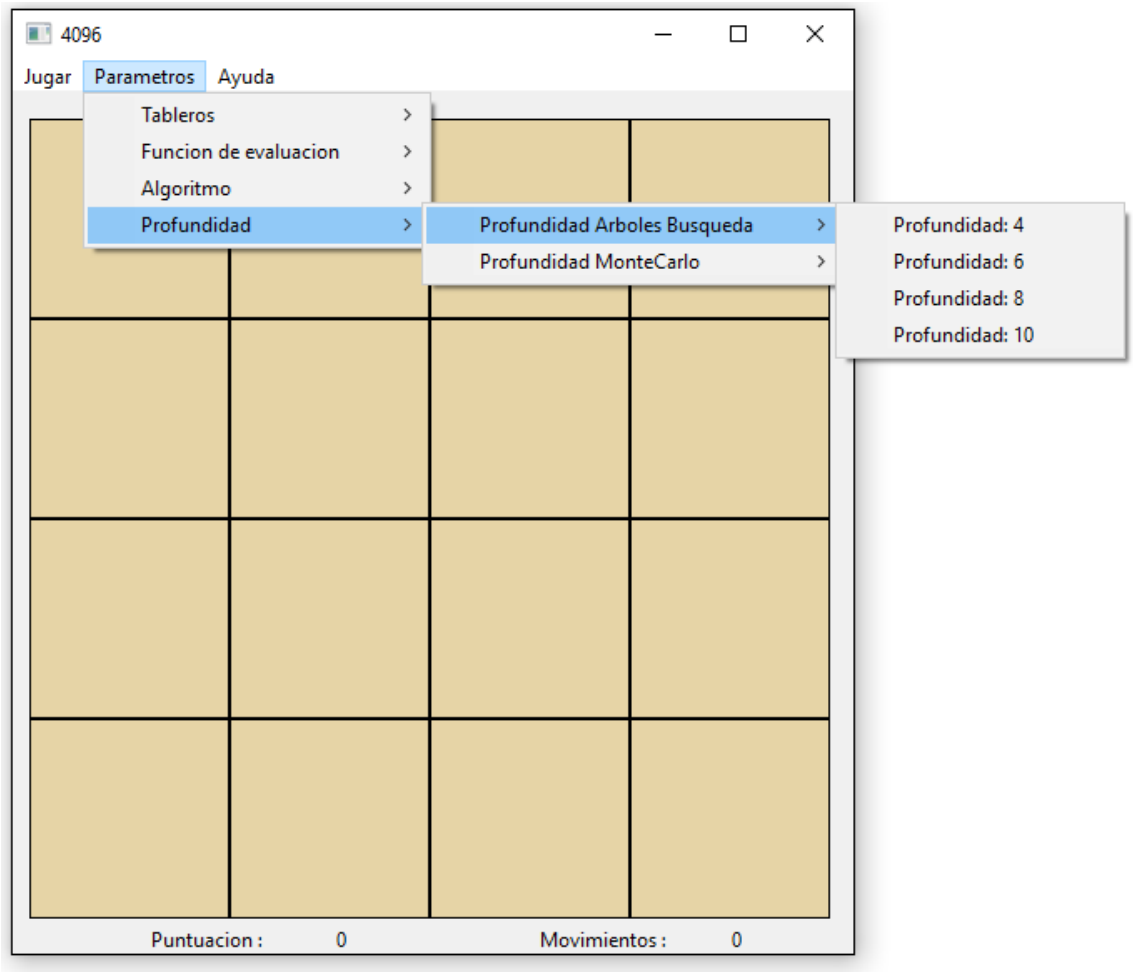


Ilustración 78 Menú profundidad búsqueda

- En caso de que el algoritmo sea el algoritmo de Monte Carlo, las posibilidades son 10, 20, 40, 60 y 80.

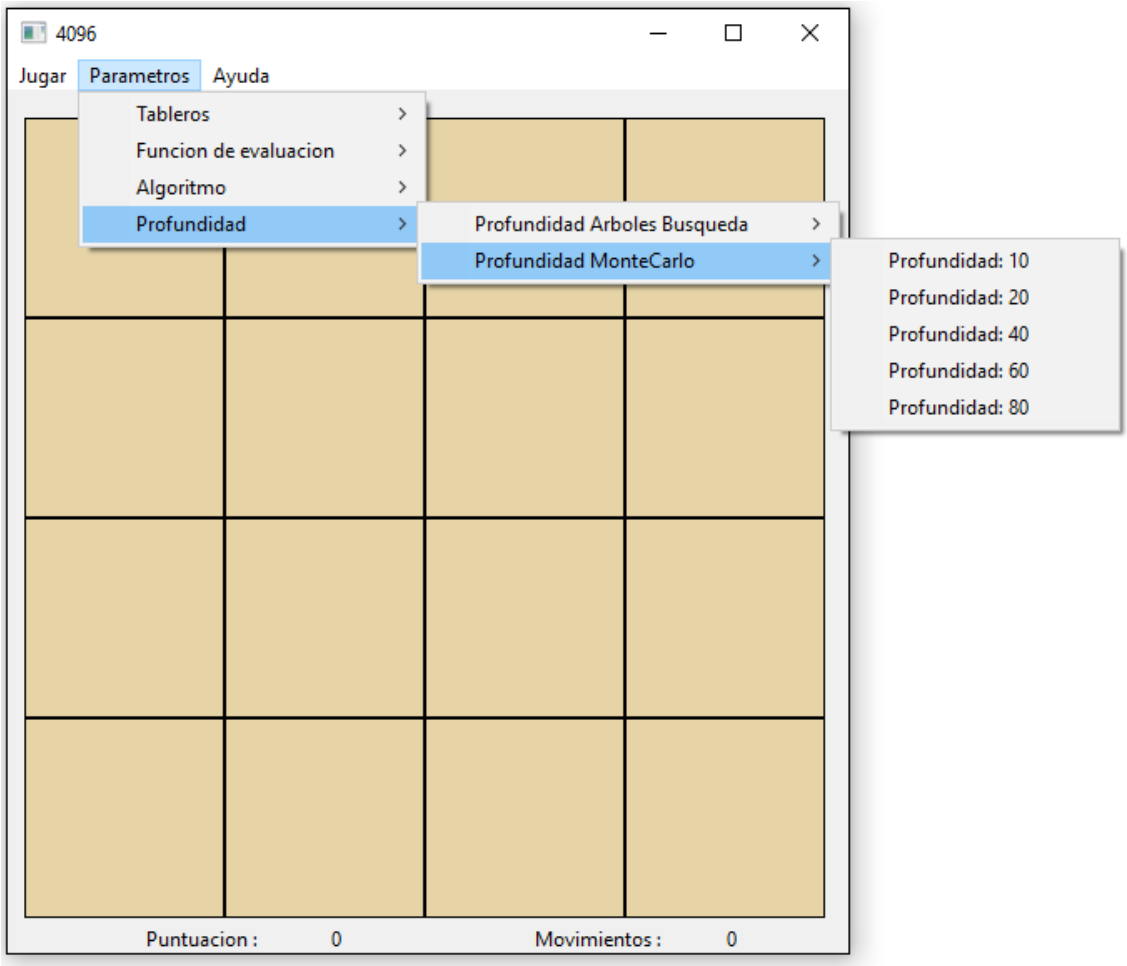


Ilustración 79 Menú profundidad Monte Carlo

El último menú, con el nombre de “Ayuda”, permitirá al usuario acceder de forma rápida a este manual para resolver cualquier duda que pueda tener.

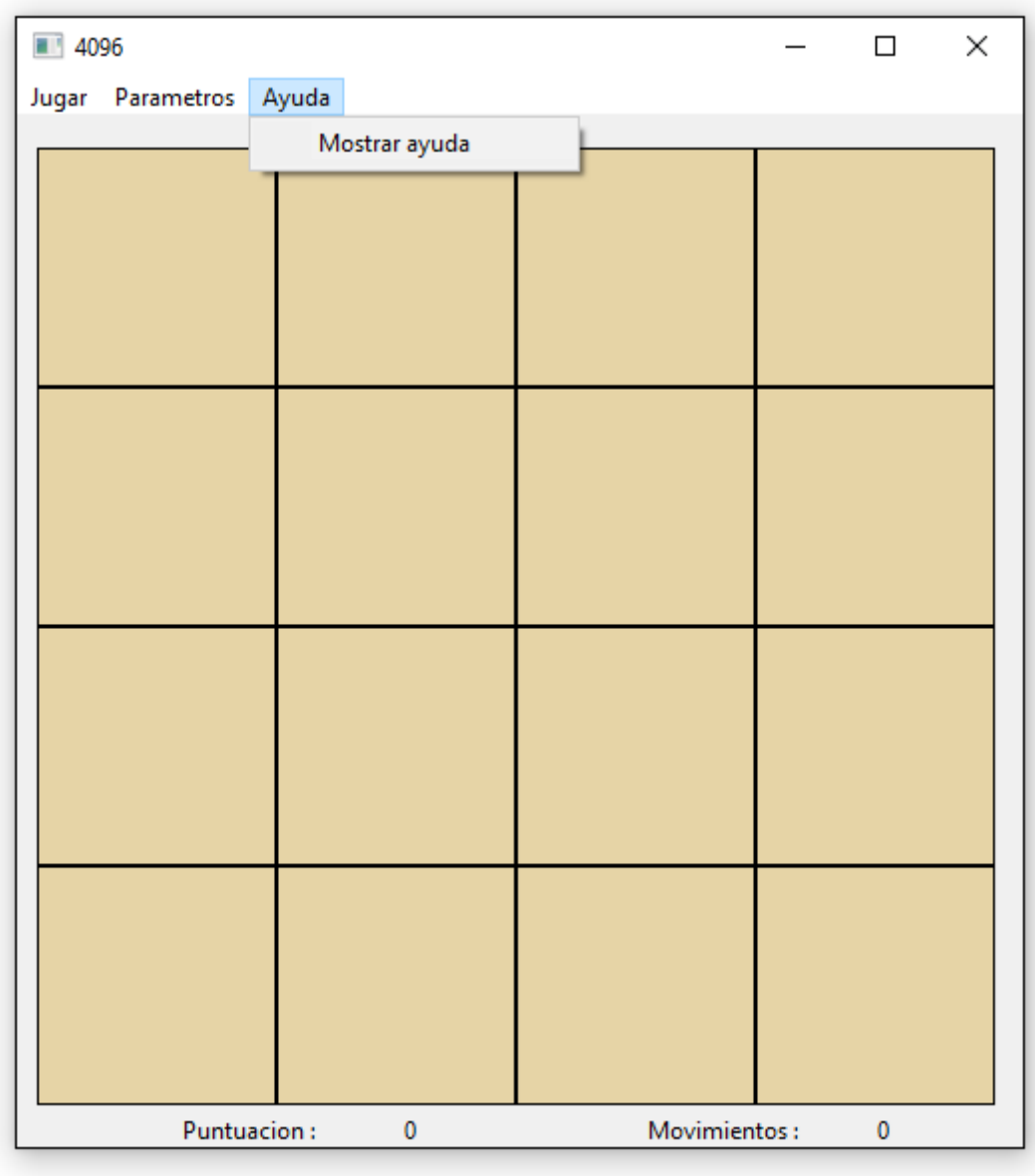


Ilustración 80 Menú ayuda

Anexo II
Pseudocódigo

A continuación, se mostrará el pseudocódigo de las principales funciones utilizadas en este proyecto:

```
Evaluar(tablero):

    tablero_expandido=Expandir_tablero(tablero)

    for i in tablero_expandido=:
        Evaluaciones=evaluar(tablero_expandido(i))

    return max(Evaluaciones)
```

Ilustración 81 Pseudocódigo Evaluar

```
Evaluar_Avanzado(tablero):

    tablero_expandido=Expandir_tablero(tablero)

    for i in tablero_expandido:
        Evaluaciones=reglas(tablero_expandido(i))

    return max(Evaluaciones)
```

Ilustración 82 Pseudocódigo Evaluar_Avanzado

```
Evaluar_Arbol(tablero):

    tablero_expandido=Expandir_tablero(tablero)

    for i in tablero_expandido:
        tablero_expandido_aux=Expandir_tablero(tablero_expandido(i)):
        Evaluaciones_aux=0

        for j tablero_expandido_aux:
            Evaluaciones_aux=evaluar(tablero_expandido(i))

        Evaluaciones=max(Evaluaciones_aux)

    return max(Evaluaciones)
```

Ilustración 83 Pseudocódigo Evaluar_Arbol

```
Busqueda_realista(tablero, profundidad, turno):
    if profundidad==0:
        return evaluacion(tablero)
    else:
        if turno==1:
            tablero_expandido=Expandir_tablero(tablero)
        else:
            tablero_expandido=Expandir_tablero_ale(tablero)
            numero_valores=len(tablero_expandido)

        for i in tablero_expandido:
            if turno==1:
                puntuacion=busqueda_realista(tablero_expandido[i], profundidad-1, turno=2)
                aux=max(puntuacion, aux)

            else:
                puntuacion=busqueda_realista(tablero_expandido[i], profundidad-1, turno=1)
                aux+=puntuacion

        if turno==2:
            aux=aux/numero_valores

        return aux
```

Ilustración 84 Pseudocódigo Busqueda_realista

```
Busqueda_pesimista(tablero, profundidad, turno):
    if profundidad==0:
        return evaluacion(tablero)
    else:
        if turno==1:
            tablero_expandido=Expandir_tablero(tablero)
        else:
            tablero_expandido=Expandir_tablero_ale(tablero)

        for i in tablero_expandido:
            if turno==1:
                puntuacion=busqueda_pesimista(tablero_expandido[i], profundidad-1, turno=2)
                alfa=max(alfa, puntuacion)
                if alfa >= beta:
                    return alfa

            else:
                puntuacion=busqueda_pesimista(tablero_expandido[i], profundidad-1, turno=1)
                beta=min(beta, puntuacion)
                if beta <= alfa:
                    return beta

        return aux
```

Ilustración 85 Pseudocódigo Busqueda_pesimista

```
Busqueda_Montecarlo(tablero):

    tablero_expandido=Expandir_tablero(tablero)
    for i in 50:
        for j in tablero_expandido:
            valor_montecarlo[j]+=simulacion_montecarlo(tablero_expandido[j])
            veces[j]+=1

    while tiempo_ejecucion<1:
        siguiente=max(valor+(C*valor_montecarlo/veces)
        valor_montecarlo[siguiente]+=simulacion_montecarlo(tablero_expandido[j])
        veces[siguiente]+=1
```

Ilustración 86 Pseudocódigo Busqueda_Monte Carlo

Anexo III Resultados de las pruebas y ejecutable del agente.

A continuación, se mostrará un enlace URL con el cual se podrá acceder a los resultados desglosados de este proyecto, así como una copia gratuita del programa implementado:

https://www.dropbox.com/sh/vbvrydu3rt2xl2n/AAAwhrj63RCvXt-WbfpelU32a?dl=0

Anexo IV
Bibliografía

[1] ABDELKADER, AHMED; ACHARYA, ADITYA; DASLER, PHILIP *On the complexity of slide-and-merge games* n.d. http://arxiv.org/pdf/1501.03837v1.pdf Último acceso: 20/04/2015

[2] ABDELKADER, AHMED; ACHARYA, ADITYA; DASLER, PHILIP *2048 is NP-complete* 4th Computational Geometry Young Researchers Forum, 2015 https://www.cs.umd.edu/~akader/files/CGYRF15_2048.pdf Último acceso: 20/04/2015

[3] BELTRACCHI, RODRIGO OSCAR, DAHL, JUAN RICARDO, RIZZALLI, AYELÉN ANALÍA *"2048 Solution": Algoritmos eficientes para la resolución del juego 2048* Universidad Nacional del Centro de la Provincia de Buenos Aires 2015 http://44jaiio.sadio.org.ar/sites/default/files/est149-166.pdf Último acceso: 20/04/2015

[4] BRADBERRY, JEFF *Introduction to Monte Carlo Tree* http://jeffbradberry.com 2015 http://jeffbradberry.com/posts/2015/09/intro-to-monte-carlo-tree-search/ Último acceso: 20/04/2015

[5] CHEN, CHRISTOPHER *2048 is in NP* http://blog.openendings.net/ 2014 http://blog.openendings.net/2014/03/2048-is-in-np.html Último acceso: 20/04/2015

[6] CIRULLI, GABRIELE *2048*, http://gabrielecirulli.github.io/2048/, Último acceso: 20/04/2015

[7] KOCSIS, LEVENTE; SZEPESVÁRI, CSABA *Bandit based Monte-Carlo Planning* Computer and Automation Research Institute of the Hungarian Academy of Sciences, n.d. http://www.sztaki.hu/~szcsaba/papers/ecml06.pdf Último acceso: 20/04/2015

[8] MEHTA, RAHUL *2048 is (PSPACE) hard, but sometimes easy* Princeton University 2014 http://arxiv.org/pdf/1408.6315v1.pdf Último acceso: 20/04/2015

[9] OLSON, RANDAL *Artificial Intelligence has crushed all human records in 2048. Here's how the AI pulled it off* http://www.randalolson.com 2014 http://www.randalolson.com/2015/04/27/artificial-intelligence-has-crushed-all-human-records-in-2048-heres-how-the-ai-pulled-it-off/ Último acceso: 20/04/2015

[10] RUSSELL, STUART; NORVIG, PETER, *Artificial Intelligence, A Modern Approach* Nueva Jersey: Ed. Prentice Hall 1995 pp 123-133

[11] WINANDS, MARK H.M; BJÖRNSSON, YNGVI; SAITO, JAHN-TAKESHI *Monte-Carlo Tree Search Solver* Games and AI Group, n.d. http://www.ru.is/faculty/yngvi/pdf/WinandsBS08.pdf Último acceso: 20/04/2015

[12] *2048 Controller Competition* 2015 http://www.cs.put.poznan.pl/wjaskowski/game-2048-competition-gecco-2015 Último acceso: 20/04/2015

[13] *2048 World Championship* https://play.google.com/store/apps/details?id=com.appgate.android.z048.worldchampionship.google.global Último acceso: 20/04/2015

Anexo V
Summary

1. Introduction

2048 is a mathematical puzzle which consists of put together equal tiles to get another one with double value. The Classic Board has 4 rows of 4 squares each, 16 in total.

At the beginning of the play, the game places two tiles 2 in two random squares. Starting from there, the player can make a move, either up, down, left or right (diagonal moves are not allowed). This movement would displace all the tiles in that direction. If any of them meets with an equal tile, both will disappear leaving in their place a new tile with double value. I.e., putting together two tiles valued in 2, will be formed a 4; putting together two 8, 16; etc.

Once the tiles have been displaced and gathered where they had to do it, the game would put another tile 2 in a random empty square. In some versions of the game it is possible to place a value 4 tile, but in this paper we will not contemplate this possibility.

The objective of the game is to get the highest possible score. This score is achieved getting together tiles, adding the value of the obtained tile. Therefore to get higher scores, it is necessary to collect tiles of higher values. Initial form, has proposed getting a value 2048 tile, which gives the name to the game. Even so, you are able to get a tile of a higher value.

The game ends when, if after a move and the corresponding random tile, the board is full and is not possible to make other moves.

2. Program

To make the relevant test, has been made a program with a very simple graphic interface. As you can see, it is a single window with a classic board. At the bottom you can see the score and the moves of the current game. At the top there are three menus, which are explained below:

- Play: with this button, a menu will appear with two options.
  - Play manual: a new game will begin. The commands for the same will be 'w' or ↑ to make a move up, "s" or ↓ to make a move down, "a" or ← to make a move left and "d" or → to make a move right. In the same way, if the "I" key is pressed, the program will execute a move based on the specified parameters.
  - Play algorithm: with this option, the program will start to play automatically according to the entered parameters. A window indicating the number of the test that is running at that time, as well as the elapsed time, estimated time and the remaining time. When you finish a game, the data will be passed to a text file called "Data.txt".

- Parameters: this is the menu that allows you to change the parameters to run the tests. It is divided into several sub-menus.
  - Boards: here it would indicate the board about who will run the tests. It can be "3x3", "4x4" or "5x5".
  - Evaluation functions: to select the mode of assessment boards.
  - Algorithm: used to select the method of selection of successors.
  - Depth: there are algorithms that is necessary to select the depth. The possibilities are 4, 6, 8, 10 or, in the case of Monte Carlo algorithm, 10, 20, 40, 60, 80.
- Help: by clicking this menu, it will skip a window that would summarize all this section.

With this tool, will be to run the tests.

3. Pre-testing

3.1. Algorithms implemented

As a first step in the attempt to resolve the 2048 puzzle, a series of very simple algorithms have been created to evaluate the results obtained by submitting the game to certain situations. To this end, two methods are needed:

- expand (board, list): This method takes a board and its associated list (indicating the free cells) and returns the four possible successors. Each successor shall correspond to a move made on this board. Returns the four boards of the successors, as well as their associated lists, score achieved in this step and if it is an "absurd" movement (that successor is identical to his father).

- evaluate (board): This method takes a board and evaluates it. For this evaluation, weights have been placed in the cells.

  Each cell will take the item that is currently and then raise it to its weight. All these values will be added to form the evaluation. Expressed mathematically:

$$Ev=\sum cell^{weight}$$

  This method will return the value of evaluation to the board.

Anticipating that this assessment would not be enough because it maximizes the evaluation with one step in advance, two other evaluation methods were created in order to have a higher score in the short-medium term.

- avanced_evaluate (board): A board shall as the previous times will be taken. When evaluating, are considered to be better the tiles that have one below exactly the same. This favors, as it passed the game, same items go together to later reunite and get a greater. This will be achieved simply by multiplying by 2 these tiles. Therefore, this algorithm would consist of:
    - Take each tile of the board.
    - Compare it with the one which is below.
    - If they are not equal, to evaluate exactly as in the previous algorithm.
    - If they are the same, multiply by two the chip and evaluate.
    - Add all evaluations to obtain the total value.

    With the above method, the evaluation priories short-term, while this would choose the successor searching the possible moves to get better tiles.

- tree (board, list): Another way of getting successors in the medium term is to take a board and its list to see which of their successors is the best. We follow these steps:
    - The board evaluated so many times as movements there.
    - Expands the board.
    - Adding the evaluation of each successor with the father.
    - Returned the best score.

    This method is similar to a tree search with depth two. However, there is no account random, so it is possible that this evaluation did not get the expected result.

3.2. Tests

Once finished the algorithms, it began to massively exploit each of them for results. 1000 tested were made with each and tests that select random moves. The following results were obtained:

Best tile	Random	%	Evaluate	%	Advanced evaluate	%	Tree	%
<=256	999	99,9	667	66,7	88	8,8	86	8,6
512	1	0,1	294	29,4	236	23,6	174	17,4
1024	0	0	39	3,9	463	46,3	441	44,1
2048	0	0	0	0	207	20,7	292	29,2
4096	0	0	0	0	6	0,6	7	0,7
Average score	1.243,28		4.469,16		15.251,732		17.246,724	
Average moves	137,863		359,495		941		1.043,82	
Average time	0,003676999		0,047796		0,134674		0,678363	
Time variance	3,24E-05		0,000503524		0,004521848		0,112571129	

Pre-testing

3.3. Conclusions

To begin with, it can be seen that all algorithms achieved a far superior to random player score. This indicates that it is a first step to solving the problem.

In addition, the last two algorithms get very superior to the simple results. Also noteworthy is that among these algorithms, the second achieved big chips with greater frequency, if it is true that the time taken is much higher.

Even so, bearing in mind that time per test is one second, it was decided to exploit stronger algorithms based on trees to get the objectives. Furthermore, rules-based algorithms optimization require much effort because of the own rules that should be designed, while those based on trees do not.

4. Tests with search trees

As discussed in the previous section, it was decided to perform more exhaustive tests using search trees. This data structure is going to make decisions based on a group of consecutive moves, making that prediction could be made in longer term. Some algorithms with different uses that pretend to approach the problem from different points of view will be used.

4.1. Evaluation functions

For starters, there have been some new evaluation functions for evaluating these trees. This is due to the decisions taken by the program are longer term, therefore it is possible to reduce the information provided by the evaluation function.

The following functions have been tested:

- "Snake": Is that used in previous tests. So-called because of the resemblance to a snake. The evaluation function value recalculates as last time.

  Where the weight is the value of each box and box the value of the chip that is in the own box.

- "Corner": In this case, it has replaced the table of weights to give more weight to the four corners.
- "Score": In this case, the evaluation function value correspond only with the accumulated score.

As shown, the information transmitted by the evaluation function decreases, with the "snake" the most informed and the "score" the least.

4.2. Search Types

Once defined the types of evaluation functions designed, it will proceed to explain searching techniques used. Both are based on the "Minimax" algorithm, but they have different approaches.

The "Minimax" algorithm is a recursive algorithm used in games with opponents to minimize the loss in the rival turn and maximize profit in our turn. It's an algorithm of depth, which means that evaluates a first node until the requested depth and then would evaluate the rest.

While it is true that the "2048" has no opponent, it has focused from the point of view of the player fighting chance. Therefore, when minimize losses, is seeking to predict which box will be selected randomly to take it into account in the decisions of the moves.

It would work like this:

- Expand the first node, which would correspond to a player's movement. It would progress up to the requested depth (always correspond to a random move).
- Once there, the leaf nodes are evaluated in terms of evaluation functions assigned and goes up the tree. Depending on the turn in question:
  - If a player's turn, all movements are evaluated. The highest value is taken.
  - If it is the turn of random, every possibility is evaluated, i.e., a new chip of value 2 in each empty position. The smallest value is taken.
- After examining the four movements, will be chosen the greatest as the winner.

It will proceed to explain each of the search types.

4.2.1. Realistic search

The "Minimax" algorithm can be improved by a pruning called "α-β" which will be shown in point 5.2.2. However, this type of search is too cautious, which often cause to not get good scores because of the lack of risk. Therefore, it was decided to slightly modify this algorithm to perform what is called "search with risk" or "Realistic search".

As already mentioned, the turn to be minimized is the assigned to random. To minimize the loss in this sense a very cautious search, because it always assumes that chance put a chip in the worst possible place. However, that is not true most of the time. Considering this issue, it proceeded to make a modification of the "Minimax" algorithm to run more risks in order to achieve better scores. This change is simply change the minimization of turn of chance to realize an average.

With this modification of the algorithm realized, we had to run.

Before trying it, there have been some optimizations of the game to adapt to using these search trees.

The first of these changes is the summary of the nodes generated by random. After a detailed observation of these nodes, it was discovered that there was redundancy that should be eliminated to speed up the evaluation.

Therefore, it was created an algorithm that summarizes the produced chips by eliminating redundancy. The performance of this algorithm is as follows:

- Empty boxes that surround that are filled are placed in a list.
- Looking for on that list right angles to the possible "triangles" and right angles are eliminated:
- Edges of the completely empty rows or columns are introduced on the same list and corners as well.

In this way, about one third of the possibilities of chance are removed, resulting in an improved exponential of time as we increase the depth.

Due to this change, it is no longer possible to find the previous average. This is because all nodes of random are not analyzed, and the summary made above, a node has more relevance than others. Therefore, it was necessary to make an approximation of the average.

The method for this purpose is made while the summary, adding a weight to each resulting node. When finding the average, each successor would be multiplied by their weight and would add to the rest. Subsequently, it is divided by the sum of the weights, giving the approximate value to be used for decision making.

Once explained the operation of this algorithm, we turn to pass the results of the test. Due to the large amount of time required for each, the number of test was reduced from 1000 to 100:

Depth 4			
Best tile	Corner %	Score %	Snake %
<=256	90	8	1
512	10	53	20
1024	0	39	46
2048	0	0	32
4096	0	0	1
Average score	2.831,8	9.314,64	19.384,24
Average moves	256,83	635,83	1.154,06
Average time	1,62249	5,76408	11,39186
Time variance	0,14883293	2,36603466	21,8661454

Realistic search depth 4

Depth 6			
Best tile	Corner %	Score %	Snake %
<=256	93	1	0
512	7	14	9
1024	5	58	31
2048	0	27	49
4096	0	0	11
Average score	3.112,16	17.745,52	27.592,28
Average moves	278,3	1.080,14	1.534,53
Average time	31,17926	186,59502	275,31277
Time variance	43,4154683	2.540,40358	13.774,1432

Realistic search depth 6

Depth 8			
Best tile	Corner %	Score %	Snake %
<=256	54	0	1
512	42	0	3
1024	4	22	16
2048	0	72	53
4096	0	6	27
Average score	5.217,04	30.259,08	36.647,88
Average moves	406,87	1.688,63	1.958,5
Average time	769,8453	5.481,93914	6.326,96956
Time variance	23.146,7335	1.345.336,66	5.059.811,36

Realistic search depth 8

First, you should discuss the bad results obtained by “corner” evaluation function. Even with 8 depth has not gotten a single game with a chip greater than 1024. The rest of evaluation functions have succeeded at that depth values of 78% for the “score” and 80% for the “snake”.

With respect to the other two evaluation functions, you can see that the evaluation function “snake” has more success rate, more scoring and more moves. However, evaluation functions “score” has achieved a success rate of 78% with 8 depth. This suggests that if we reinforce this evaluation function with a minimum of additional information, it is possible that the results improve very much.

In addition, the latter evaluation function is very close in terms of scoring in the last depth. Compared with the amount of information they need to get the results, the evaluation functions “score” has much more merit, since with a minimum of information almost gets at the height of the more informed evaluation function.

4.2.2. Pessimistic search

As explained in the previous section, an improved version of the “Minimax” algorithm was also used. This improvement is “α-β” pruning, that manages to eliminate redundant nodes without having to evaluate them, ensuring that the final result will not be altered. This is a great improvement of time, which allows to run tests at great depths with a reasonable time.

This pruning is to be updating the values of the “α” and “β” variables, each being the best and the worst option so far. Now we will proceed to explain its operation:

- The first node to its leaf is evaluated in depth. The value of this node is returned.
- We will evaluate the following tree branch:
    - If the parent is a maximization step, which means the child is one of minimization. The “α” value will be equal to the highest value previously evaluated, while the “β” will be updated. If at some point “β” becomes less than or equal to “α”, the remaining nodes will not be necessary, since the resulting value is always less than α, therefore, this branch of the tree will never be the winner.
    - Otherwise, something like that would happen. If “β” is the father and there is a “α” greater than or equal to it, it will deal a redundant branch and its evaluation is not needed.

Using this improvement, it comes with 4, 6 and 8 depth testing. It is necessary to indicate that the summary of possibilities of chance that was explained in the previous section has also used.

Depth 4			
Best tile	Corner %	Score %	Snake %
<=256	95	12	1
512	5	60	14
1024	0	28	29
2048	0	0	43
4096	0	0	13
8192	0	0	0
Average score	2.672,88	8.205,8	27.847,16
Average moves	249,52	573,26	1.563,47
Average time	1,25401	3,17109	11,91442
Time variance	0,11613086	0,73436042	33,160554

Pessimistic search depth 4

Depth 6			
Best tile	Corner %	Score %	Snake %
<=256	91	2	2
512	9	29	3
1024	0	64	18
2048	0	5	42
4096	0	0	34
8192	0	0	1
Average score	3.287,88	13.145,04	42.188
Average moves	289,75	846,06	2.239,23
Average time	13,82228	39,75809	189,3512
Time variance	12,8870315	104,905492	6.915,741

Pessimistic search depth 6

Depth 8			
Best tile	Corner %	Score %	Snake %
<=256	83	1	0
512	17	15	0
1024	0	64	11
2048	0	20	44
4096	0	0	39
8192	0	0	6
Average score	4.103,2	17.206,12	51.142,8
Average moves	343,35	1.056,17	2.632,53
Average time	120,77807	378,36625	2.112,37091
Time variance	906,298874	10.934,0338	572.011,727

Pessimistic search depth 8

As shown, the score, the movements and the time of “snake” evaluation functions are much higher than the other two. The “corner” evaluation function, even with 8 depth, has failed to reach the 2048 target even once, which is overshadowed by the 20% of the “score” and the surprising 89% of the “snake”.

The variance of time of the “snake” evaluation function is also much higher than the rest. This is because between the worst and the best match of each of these evaluation function configuration are much different than in other cases.

4.3. Conclusions

To conclude this point, the first thing that must be mentioned is that the “corner” evaluation function has not approached the expected results. In fact, it has failed to get a record of 2048 in any case, so a reform it is necessary or discarded this evaluation function.

Regarding the “score” evaluation function, it has obtained very relevant results in the Realistic search, but it has not been the case of the pessimistic search. It would be interesting to consider a modification of it to reinforce it with some information and thus achieve better results.

Finally, “snake” evaluation function has achieved great results, especially in pessimistic search. The amount of information provided to the game is more than enough to get tokens until 8192. In the case of pessimistic search 8 depth, got an average score of 51.142,8, a brand that is very difficult to overcome.

In conclusion, despite having a clear winner with “snake” evaluation function, it is not ruled out using slightly modify the “score” to try to improve it.

5. Test with Monte Carlo algorithm

Continuing with the algorithms used to solve the problem, will be to perform a non-deterministic algorithm based on Monte Carlo algorithm.

A non-deterministic algorithm is one that, under one scenario, will lead to different results. This occurs because they do not make their decisions taking into account all possibilities, but it does take a subset of them and exploits at great depths.

5.1. Algorithm implemented

This Monte Carlo algorithm has been modified significantly to suit the problem of "2048", although used as a basis for which will be described below.

The first major change is the formula with which the gain of the exploitation and exploration is calculated. Instead of using UCT, it has been used as follows:

$$x = value(x) + C\,\frac{\sum value(test)}{nº\_test}$$

The first part corresponds to exploration, and will be simply the score with respect to the base board due to this movement. The second part will deal with exploration, with an average of the values obtained in this method multiplied by the constant exploration valuation.

The algorithm works as follows:

- The board that you want to find out his best move is taken and their possible successors are expanding
- Then, a series of simulations with each of the resulting boards are made. It is played without commitment as follows:
    - A random tile is placed.
    - A movement with a predetermined criterion is selected. These criteria determine the importance given to the possible successors, giving the possibility that, although one of them seems to not have good result, can be chosen just in case one of its successors can get it. These criteria include:
        - **Probabilistic:** Each event is assigned a probability based on the score.
        - **Direct:** the successor with higher value will be selected.
        - **Random:** a random successor will be selected.
    - It will continue with this process until the maximum depth or when the player loses. Returns the values obtained in this way:

$$TotalValue += \frac{ActualDepthVaue}{ActualDepth}$$

So higher values will be obtained if you get much score at low depths. This will prioritize plays that get significant changes in the score quickly, instead of many small scores. If it has met with defeat, it will severely penalize this value.

This method repeats a predetermined number of times, and at each step the value obtained is collected and added to the corresponding movement.

5.2. Tests

Now, the results obtained with this algorithm will be show:

Depth 10			
Best tile	Corner %	Score %	Snake %
<=256	32	0	32
512	63	4	63
1024	5	23	5
2048	0	66	0
4096	0	7	0
8192	5.538,88	30.091,32	5.274,76
Average score	417,09	1.682,98	396,3
Average moves	512,79056	1.983,67816	498,85459
Average time	22.436,04458	405.626,0332	23.032,06538

Monte Carlo search depth 10

Depth 20			
Best tile	Corner %	Score %	Snake %
<=256	40	0	34
512	56	0	59
1024	4	7	7
2048	0	83	0
4096	0	10	0
8192	5.077,12	35.622,56	5.251,56
Average score	387,95	1.954,37	393,54
Average moves	554,80616	2.614,18878	583,74247
Average time	35.555,5426	330.818,013	42.503,08931

Monte Carlo search depth 20

Depth 40			
Best tile	Corner %	Score %	Snake %
<=256	24	0	41
512	69	1	56
1024	7	29	3
2048	0	69	0
4096	0	1	0
8192	5.629,6	27.305,04	4.706,72
Average score	416,12	1548,56	360,67
Average moves	750,55578	2.503,49588	666,54931
Average time	52.925,30691	366.125,1448	52.871,2903

Monte Carlo search depth 40

Depth 60			
Best tile	Corner %	Score %	Snake %
<=256	21	0	35
512	70	4	57
1024	9	42	8
2048	0	47	0
4096	0	10	0
8192	5.927,08	24.097,56	5.169,44
Average score	433,94	1394,61	384,85
Average moves	892,6719	2.455,72039	810,07291
Average time	65.144,79279	457.690,9239	84.047,6202

Monte Carlo search depth 60

Depth 80			
Best tile	Corner %	Score %	Snake %
<=256	21	0	35
512	70	4	57
1024	9	42	8
2048	0	47	0
4096	0	10	0
8192	5.927,08	24.097,56	5.169,44
Average score	433,94	1394,61	384,85
Average moves	892,6719	2.455,72039	810,07291
Average time	65.144,79279	457.690,9239	84.047,6202

Monte Carlo search depth 80

5.3. Conclusions

It is remarkable to note the failure of both the evaluation function "Snake" (which had obtained such good results in previous tests) and the evaluation function "Corner" when it comes to non-deterministic searches.

However, the other evaluation function corresponding to the score, won a surprisingly high percentage of success, surpassing all achieved so far. While your score can not overcome the rest so drastically, if the main objective was to get a piece of 2048 this is the best option.

6. Final conclusions

Although it has required much effort, it has achieved all the goals in a very satisfactory manner.

The most important problems found throughout the completion of this work have been learning and use of the programming language *Python*, as well as the *Wxpython* library. Because I have never used these tools, initially I committed many mistakes when programming the project. However, with the increasing knowledge about them, the number of errors was reduced gradually until it disappears.

Similarly, I have learned the use and implementation of powerful Artificial Intelligence algorithms as Minimax, the α-β and Monte Carlo, which are often used in any area of this fascinating discipline.

About to the personal motivation with which this project was addressed, has managed to satisfy the curiosity of whether the strategy followed was appropriate, because the evaluation function has performed better than its alternatives is what I usually use. In addition, the program often get better scores than me, which also greatly satisfies the objectives and motivation that led this project.